

**NORTHEASTERN UNIVERSITY**

**Graduate School of Engineering**

**Thesis Title:** Parallel Backprojection: A Case Study in High Performance Reconfigurable Computing

**Author:** Benjamin Cordes

**Department:** Electrical and Computer Engineering

Approved for Thesis Requirements of the Master of Science Degree

---

Thesis Advisor: Prof. Miriam Leeser

---

Date

---

Thesis Reader: Prof. Eric Miller

---

Date

---

Thesis Reader: Prof. David Kaeli

---

Date

---

Department Chair: Prof. Ali Abur

---

Date

Graduate School Notified of Acceptance:

---

Dean: Prof. Yaman Yener

---

Date

# Parallel Backprojection: A Case Study in High Performance Reconfigurable Computing

A Thesis Presented

by

**Benjamin Cordes**

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements  
for the degree of

**Master of Science**

in

Electrical Engineering

in the field of

Computer Engineering

**Northeastern University**  
**Boston, Massachusetts**

April 2008

© Copyright 2008 by Benjamin Cordes  
All Rights Reserved

# Acknowledgements

First and foremost I owe many thanks to my advisor, Dr. Miriam Leeser, for her valuable technical guidance and support behind the scenes. Without her this project simply would not have been possible.

Dr. Eric Miller made his extensive expertise with signal processing and radar algorithms available to me throughout the project, and assisted with gathering the information for Section 2.1.

Al Conti, my fellow graduate student in the Reconfigurable Computing Laboratory, was a constant source of support and good cheer around the office. His assistance on the first implementation of the backprojection project provided the foundation for the work in this document.

Finally, to my colleagues in the RCL, I express my heartfelt thanks for a friendly and enjoyable work environment and I wish them only the best in their future endeavors.

# Abstract

High-Performance Reconfigurable Computing (HPRC) is a novel approach to providing the kind of large-scale computing power demanded by modern scientific applications. One of these applications is *Backprojection*, an image formation algorithm that can be used as part of a *Synthetic Aperture Radar* (SAR) processing system. Backprojection is an embarrassingly parallel application that is ideal for implementation in hardware, especially on reconfigurable computing devices such as FPGAs.

This thesis presents an implementation of backprojection for SAR on an HPRC system. It is intended for use in the Swathbuckler project, a joint collaboration of multiple national military research agencies. Using simulated data taken at a variety of ranges, our HPRC implementation of backprojection runs over 200 times faster than a similar software-only program, with an overall application speedup of better than 50x. In addition to excellent single-node performance, the application is easily parallelizable, achieving near-linear speedup when run on multiple nodes of a clustered HPRC system.

# Contents

- 1 Introduction** **1**
  
- 2 Background** **4**
  - 2.1 Backprojection Algorithm . . . . . 4
  - 2.2 Supercomputer Architectures . . . . . 7
    - 2.2.1 HPTi, Inc. HHPC . . . . . 7
    - 2.2.2 Other HPRC Architectures . . . . . 11
  - 2.3 Related Work . . . . . 15
    - 2.3.1 Backprojection . . . . . 16
    - 2.3.2 Other Applications . . . . . 18
    - 2.3.3 Best Practices and Future Development . . . . . 20
  - 2.4 Summary . . . . . 22
  
- 3 Experimental Design** **24**
  - 3.1 System Background . . . . . 24
    - 3.1.1 Swathbuckler Project . . . . . 25
    - 3.1.2 HHPC Features . . . . . 26

3.2	Algorithm Analysis	28
3.2.1	Parallelism Analysis	28
3.2.2	Dividing The Problem	30
3.2.3	Memory Allocation	33
3.2.4	Data Formats	34
3.2.5	Strength Reduction	35
3.2.6	Data Collection Parameters	36
3.3	FPGA Design	37
3.3.1	Clock Domains	38
3.3.2	Control Registers and DMA Input	39
3.3.3	Datapath	40
3.3.4	Magnitude Computation and DMA Output	45
3.4	Software Design	46
3.4.1	Single-node software	46
3.4.2	Multi-node parallel software	48
3.5	Summary	49
<b>4</b>	<b>Experimental Results</b>	<b>50</b>
4.1	Experimental Setup	50
4.1.1	Program Structure	51
4.1.2	Test Data	52
4.2	Results and Analysis	53

4.2.1	Single Node Performance . . . . .	53
4.2.2	Single Node Accuracy . . . . .	57
4.2.3	Multi-Node Performance . . . . .	58
4.3	Summary . . . . .	60
<b>5</b>	<b>Conclusions</b>	<b>62</b>
5.1	Future Backprojection Work . . . . .	62
5.2	HPRC Systems and Applications . . . . .	63
5.3	Summary . . . . .	65
<b>A</b>	<b>Experimental Data Sets</b>	<b>66</b>

# List of Figures

2.1	The HPTi HNPC . . . . .	8
2.2	Diagram of the WildStar II FPGA Board . . . . .	9
2.3	Diagram of the Virtex II architecture . . . . .	10
3.1	Block diagram of Swathbuckler system . . . . .	25
3.2	Division of target image across multiple nodes . . . . .	31
3.3	Block diagram of backprojection hardware unit . . . . .	38
3.4	Block diagram of hardware datapath . . . . .	41
4.1	Multi-node scaling performance . . . . .	59
A.1	Target locations, dataset 1 . . . . .	68
A.2	Result image, hardware program, dataset 1 . . . . .	68
A.3	Result image, software program, dataset 1 . . . . .	68
A.4	Target locations, dataset 2 . . . . .	69
A.5	Result image, hardware program, dataset 2 . . . . .	69
A.6	Result image, software program, dataset 2 . . . . .	69
A.7	Target locations, dataset 3 . . . . .	70

A.8	Result image, hardware program, dataset 3 . . . . .	70
A.9	Result image, software program, dataset 3 . . . . .	70
A.10	Target locations, dataset 4 . . . . .	71
A.11	Result image, hardware program, dataset 4 . . . . .	71
A.12	Result image, software program, dataset 4 . . . . .	71

# List of Tables

2.1	Comparison of HPRC Architectures . . . . .	15
4.1	Single-node experimental performance . . . . .	53
4.2	Single-node backprojection performance by data set . . . . .	56
4.3	Image accuracy by data set . . . . .	57
4.4	Multi-node experimental performance . . . . .	58
4.5	Speedup factors for hardware program . . . . .	60
A.1	Data set characteristics . . . . .	67

# Chapter 1

## Introduction

In the continuing quest for computing architectures that are capable of solving ever-expanding problems, one of the newest directions of study is *High-Performance Reconfigurable Computing* (HPRC). HPRC can be defined as a marriage of traditional high-performance computing (HPC) techniques and reconfigurable computing (RC) devices.

HPC is a well-known set of architected solutions for speeding up the computation of problems that can be divided neatly into pieces. Multiple general-purpose processors (GPPs) are linked together such that they can share data, the pieces of the problem are distributed to the individual processors and computed, and then the answer is assembled from the pieces.

Reconfigurable computing uses many of the same concepts as HPC, but on a smaller scale. A special-purpose processor (SPP), often a Field-Programmable Gate Array (FPGA), is attached to a GPP and programmed to execute a useful function. Special-purpose hardware computes the answer to the problem quickly by exploiting

hardware design techniques such as pipelining, the replication of small computation units, and high-bandwidth local memories.

Both of these computing architectures reduce computation time by exploiting the *parallelism* inherent in the application. They rely on the fact that multiple parts of the overall problem can be computed relatively independently of each other. Though HPC and RC act on different levels of parallelism, in general, applications with a high degree of parallelism are well-suited for these architectures.

The idea behind HPRC is to provide a computing architecture that takes advantage of both the *coarse-grained* parallelism exploited by clustered HPC systems and the *fine-grained* parallelism exploited by RC systems. In theory, more exploited parallelism means more speedup and faster computation times. In reality, several factors may prevent performance from improving as much as is desired.

In this thesis we will examine one application that contains a very high degree of parallelism. The *backprojection* image formation algorithm for synthetic aperture radar (SAR) systems is “embarrassingly parallel”, meaning that it can be broken down and parallelized on many different levels. For this reason, we chose to implement backprojection on an HPRC machine at the Air Force Research Laboratory in Rome, NY, as part of a SAR processing system. We present an analysis of the algorithm and its inherent parallelism, and we describe the implementation process along with the design decisions and pitfalls that went into the solution.

Our contributions are:

- We implement the backprojection algorithm for SAR on an FPGA. Though backprojection has been implemented many times in the past (see Section 2.3), FPGA implementations of backprojection for SAR are not well-represented in the literature.
- We further parallelize this implementation by developing an HPC application that produces large SAR images on a multi-node HPRC system.

The rest of this thesis is organized as follows. Chapter 2 provides some background information on the backprojection algorithm and the HPRC system on which we implemented it, as well as some information on other HPRC systems and applications that have been implemented on them. Chapter 3 describes the backprojection implementation and how it fits into the overall backprojection application. The performance data and results of our design experiments are analyzed in Chapter 4. Finally, Chapter 5 draws conclusions and suggests future directions for research.

# Chapter 2

## Background

This chapter provides supporting information that is useful to understanding the application presented in Chapter 3. Section 2.1 describes Backprojection and SAR, highlighting the mathematical function that we implemented in hardware. Section 2.2 gives more details about HPRC systems that have been constructed, including the one which hosts our application. Section 2.3 provides a review of the current literature on the subjects of backprojection and HPRC systems.

### 2.1 Backprojection Algorithm

We briefly describe the backprojection algorithm in this section. Further details on the radar processing and signal processing aspects of this process can be found in [51, 29].

Backprojection is an image reconstruction algorithm that is used in a number of applications, including medical imaging (Computed Axial Tomography, or CAT) and Synthetic Aperture Radar (SAR). The implementation described herein is used in a SAR application. For both radar processing and medical imaging applications,

backprojection provides a method for reconstructing an image from the data that are collected by the transceiver.

SAR data are essentially a series of time-indexed radar reflections observed by a single transceiver. At each step along the synthetic aperture, or flight path, a pulse is emitted from the source. This pulse reflects off elements in the scene to varying degrees, and is received by the transceiver. The observed response to a radar pulse is known as a “trace”.

SAR data can be collected in one of two modes, “stripmap” and “spotlight”. These modes describe the motion of the radar relative to the area being imaged. In the spotlight mode of SAR, the radar circles around the scene. Our application implements the stripmap mode of SAR, in which radar travels along a straight and level path.

Regardless of mode, given a known speed at which the radar pulse travels, the information from the series of time-indexed reflections can be used to identify points of high reflectivity in the target area. By processing multiple traces instead of just one, a larger radar aperture is synthesized and thus a higher-resolution image can be formed.

The backprojection image formation algorithm has two parts. First, the radar traces are filtered according to a linear time-invariant system. This filter accounts for the fact that the airplane on which the radar dish is situated does not fly in a perfectly level and perfectly straight path. Second, after filtering, the traces are

“smeared” across an image plane along contours that are defined by the SAR mode; in our case, the flight path of the plane carrying the radar. Coherently summing each of the projected images provides the final reconstructed version of the scene.

Backprojection is a highly effective method of processing SAR images, but it is computationally complex compared to traditional Fourier-based image formation techniques. This has made backprojection prohibitively difficult to implement on traditional computers to date. However, the smearing process contains a high degree of parallelism, which makes it suitable for implementation on reconfigurable devices.

The operation of backprojection takes the form of a mapping from projection data  $p(t, u)$  to an image  $f(x, y)$ . A single pixel of the image  $f$  corresponds to an area of ground containing some number of objects which reflect radar to a certain degree. Mathematically, this relationship is written as:

$$f(x, y) = \int p(i(x, y, u), u) du \quad (2.1)$$

where  $i(x, y, u)$  is an indexing function indicating, at a given  $u$ , those  $t$  that play a role in the value of the image at location  $(x, y)$ . For the case of SAR imaging, the projection data  $p(t, u)$  take the form of the filtered radar traces described above. Thus, the variable  $u$  corresponds to the slow-time<sup>1</sup> location of the radar, and  $t$  is the fast-time index into that projection. The indexing function  $i$  takes the following form for SAR:

$$i(x, y, u) = \chi(y - x \tan \phi, y + x \tan \phi) \cdot \frac{\sqrt{x^2 + (y - u)^2}}{c} \quad (2.2)$$

---

<sup>1</sup>Fast-time variables are related to the speed of radar propagation (i.e. the speed of light), while slow-time variables are related to the speed of the airplane carrying the radar.

with  $c$  the speed of light,  $\phi$  the beamwidth of the radar, and  $\chi(a, b)$  equal to 1 for  $a \leq u \leq b$  and 0 otherwise.  $x$  and  $y$  describe the two-dimensional offset between the radar and the physical spot on the ground corresponding to a pixel, and can thus be used in a simple distance calculation as seen in the right-hand side of Equation (2.2).

In terms of implementation, we work with a discretized form of Equation (2.1) in which the integral is approximated as a Riemann sum over a finite collection of projections  $u_k$ ,  $k \in \{1 \dots K\}$  and is evaluated at the centers of image pixels  $(x_i, y_j)$ ,  $i \in \{1 \dots N\}, j \in \{1 \dots K\}$ . Because evaluation of the index function at these discrete points will generally not result in a value of  $t$  which is exactly at a sample location, interpolation can optionally be performed in the variable  $t$  to increase accuracy.

## 2.2 Supercomputer Architectures

In this section we describe some existing and soon-to-be-released HPRC systems. We begin with a detailed description of the system on which we implemented the backprojection algorithm from the previous section, and proceed with an overview of other important HPRC systems.

### 2.2.1 HPTi, Inc. HHPC

This project aimed at exploiting the full range of resources available on the Heterogeneous High Performance Cluster (HHPC) at the Air Force Research Laboratory in Rome, NY[44]. Built by HPTi[26], the HHPC (seen in Figure 2.1) features a Beowulf



Figure 2.1: The HPTi HHPC  
From [44]

cluster of 48 heterogeneous computing nodes, where each node consists of a dual 2.2GHz Xeon PC running Linux and an Annapolis Microsystems WildStar II FPGA board.

Figure 2.2 shows a block diagram of the WildStar II FPGA board. It features two VirtexII FPGAs and connects to the host Xeon general-purpose processors (GPPs) via the PCI bus. Each FPGA has access to 6MB of SRAM, divided into 6 banks of 1MB each, and a single 64MB SDRAM bank. The Annapolis API supports a master-slave paradigm for control and data transfer between the GPPs and the FPGAs. Applications for the FPGA can be designed either through traditional HDL-based design and synthesis tools, or by using Annapolis's CoreFire[3] module-based design suite.

A block diagram of the components of the VirtexII FPGA can be seen in Fig-

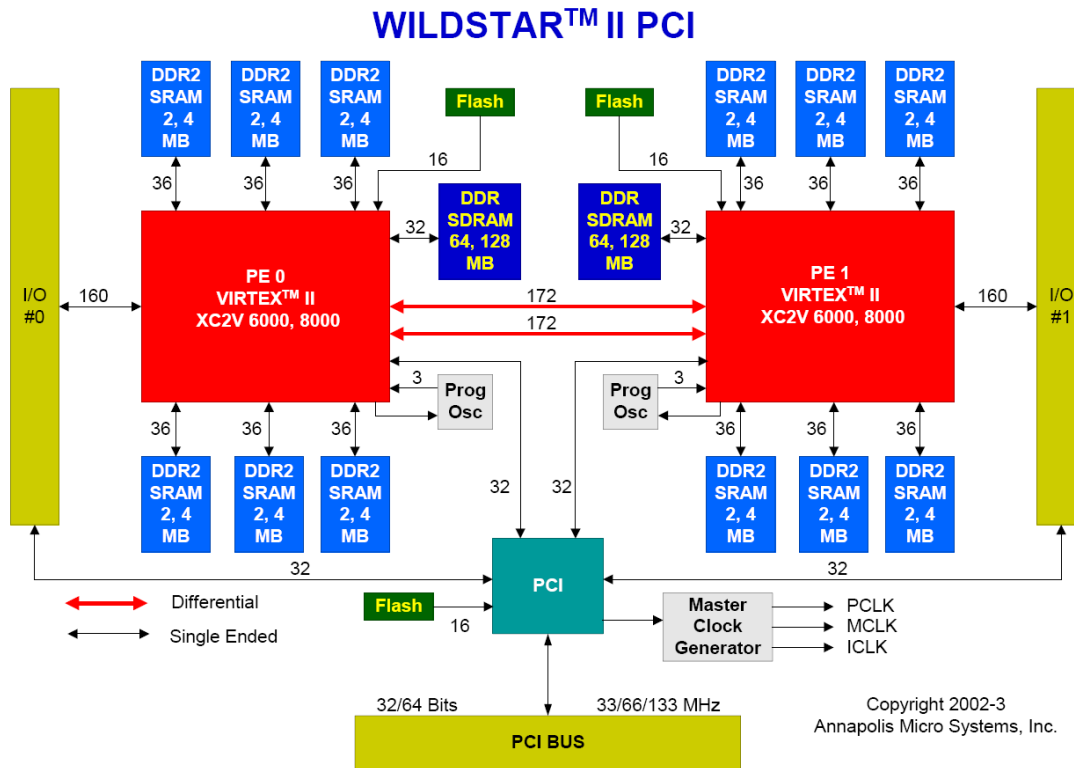


Figure 2.2: Diagram of the WildStar II FPGA Board  
From [4], ©2001-2003 Annapolis Microsystems, Inc.

ure 2.3. Like all Xilinx FPGAs it is primarily composed of Configurable Logic Blocks (CLBs) that can be programmed to compute a wide variety of logic functions. The VirtexIIs also feature several embedded multipliers, which can be more efficient than using CLBs to implement an integer multiply. The final noteworthy components are the Block SelectRAMs distributed across the chip. These configurable dual-ported memories are very useful for storing data close to the logic, providing fast access times and often high bandwidth.

The nodes of the HHPc are linked together in three ways. The PCs are directly connected via gigabit Ethernet as well as through Myrinet MPI cards. The WildStar

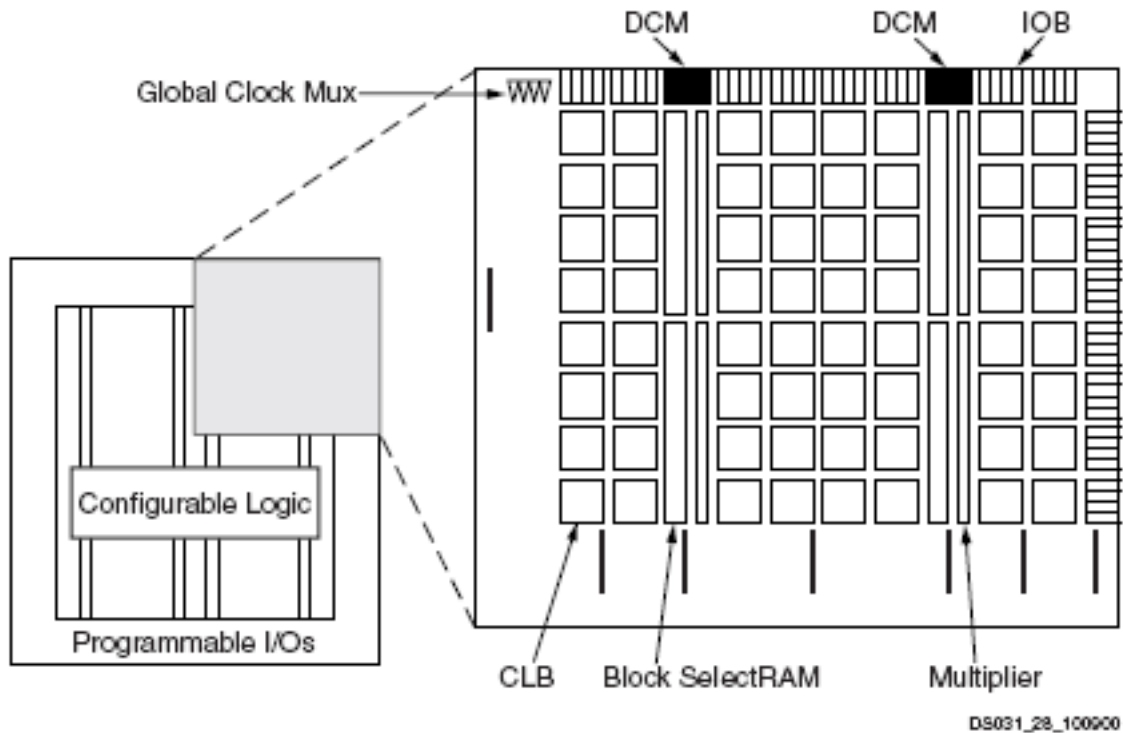


Figure 2.3: Diagram of the Virtex II architecture  
From [59], ©2000-2006 Xilinx, Inc.

II boards are also directly connected to each other through an LVDS (low-voltage differential signalling) I/O daughtercard, which provides a systolic interface over which each FPGA board may talk to its nearest neighbor in a ring. Communication over Ethernet is supplied by the standard C library under Linux. Communication over Myrinet is achieved with an installation of the MPI message-passing standard, though MPI can also be directed to use Ethernet instead. Communicating through the LVDS interconnect involves writing communications modules for the FPGA manually.

This architecture represents perhaps the most direct method for adding reconfigurable resources to a supercomputing cluster. Each node's architecture is similar to

that of a single-node reconfigurable computing solution. Networking hardware which interfaces well to the Linux PCs is included to create the cluster network. The ability to communicate between FPGAs is included but remains difficult for the developer to employ.

### 2.2.2 Other HPRC Architectures

Supercomputing platforms with reconfigurable hardware are emerging with very different architectures, each with their own programming paradigm. Some systems employ a master-slave model where an FPGA is coupled with a microprocessor. Other systems use FPGAs as independent computing elements. Communication is supported on some systems through shared memory, while others rely on messages passed over a distributed memory architecture. The architecture of a system directly affects the performance of applications running on that system, so herein we compare the architectures of existing HPRC systems.

The XD1 was Cray's[15] first supercomputer with FPGA coprocessors. The XD1 is a scalable cluster of AMD Opteron processors and VirtexII-Pro FPGAs. An XD1 chassis houses six motherboards. Each board contains two Opterons, two RapidArray processors to manage communication, an FPGA for application acceleration, banks of memory dedicated to each of the two Opterons and the Virtex FPGA, and a disk for storage. An all-to-all switch connects all twelve RapidArray processors in a single chassis.

The architecture of the XD1 provides for a microprocessor-centric programming

paradigm. Each FPGA acts as a slave to one of the CPUs on the motherboard, which is a familiar model for most programmers. Several third party tools[10, 17, 28, 38] target the XD1 architecture and address the problems of hardware/software co-design from a programming level which is familiar to users with a background in sequential software. However, efficient utilization of the XD1 currently requires knowledge of HDL programming as well as experience with hardware/software co-design and parallel programming.

Though the XD1 has been discontinued, Cray announced in late 2007 a follow-on product that will feature a similar architecture but updated hardware. This machine is called the *XT5<sub>h</sub>* and is an optional configuration of the XT5 supercomputer[16]. The ‘h’ in the product name indicates a ‘hybrid’ configuration, meaning that there are both traditional GPP processing nodes as well as FPGAs.

Mercury Computer Systems offers several high performance systems that feature reconfigurable computing devices. The 6U VME system[36] features a RACEway backplane into which daughtercards are connected. Each daughtercard contains either PPC general-purpose processors (GPPs) or Mercury FPGA Compute Nodes (FCNs)[32]. Owing to the embedded focus of the 6U VME system, Mercury provides the Multi-Computer Operating Environment[33] which allows programs to be run on distributed systems with a smaller memory footprint and without the overhead of a traditional OS. Users also have the option of running the Linux kernel on the 6U system. Mercury also offers similar capabilities in their Powerstream line of

products[35, 34].

Unlike the XD1, Mercury FCNs do not require a general purpose processor to load and store data. FCNs are standalone processing elements with the ability to load and store data to and from any device connected to the RACEway fabric. However, the high performance and versatility of Mercury systems comes at the cost of custom development at every stage of the development process. Mercury does provide their own FCN Development Kit[31] for application design.

The SRC-7[53] is SRC's latest HPRC system. It is similar to the Mercury products in that nodes with FPGAs can function independently without direct control from a GPP. The SRC-7 can also be configured in a flexible fashion, much like the Mercury system. SRC-7 nodes comprise either a GPP running Linux, a reconfigurable MAP processor, or a large block of standalone memory. A MAP node is made up of two VirtexII FPGAs that communicate via shared memory and direct connectors. Daughtercards are connected an all-to-all backplane switch called "Hi-Bar". In addition to the Hi-Bar switch, general purpose I/O connectors are available which can be used either for external I/O or to directly connect the MAP processors to each other.

SRC's Carte programming environment[52] utilizes many advanced techniques in reconfigurable application development. Source code can be written in Fortran and/or C. The Carte compiler observes the computational network of MAP processors and microprocessors that are available. The MAP compiler extracts parallelism

from the source code and generates hardware pipelines to be instantiated on the MAP processors. The remaining code is compiled to produce serial code for the microprocessors. Carte then assembles all of the compiled code together into a single executable.

SGI[48] introduced RASC (Reconfigurable Application-Specific Computing) blades as an addition to their line of HPC servers. Each RASC blade is composed of a pair of Xilinx Virtex4 FPGAs, an interface to the NUMalink backplane, and on-board SRAM. NUMalink allows the RASC blades to be seamlessly connected to a controlling cluster of general-purpose processors such as the SGI Altix. All blades have access to globally shared memory. The included API supports interfacing between the GPPs and the FPGAs using a master-slave paradigm. The programming of the reconfigurable hardware in a RASC blade follows the same model as the XD1: configurations must be custom, but third-party tools are available for automated generation of hardware components from a higher level programming interface.

For comparison, we now summarize our discussion of the HPTi[26] Beowulf cluster from the previous section. It features a fixed cluster of 48 compute nodes. Each node consists of a pair of Xeon GPPs and an Annapolis Microsystems WildStar II PCI board with two VirtexII FPGAs. Node-to-node communication is accomplished over a Myrinet MPI network as well as standard Gigabit Ethernet. The Annapolis API supports a master-slave paradigm for control and data transfer between the GPPs and the FPGAs. Applications for the FPGA can be designed either through traditional

Supercomputing Architecture	Interconnect Backplane	Operating System	Communication Paradigm	Programming Environment
Cray XD1	RapidArray	Linux	Master-Slave	various
Mercury 6U VME SRC-7	RACEway Hi-Bar	MCOE Linux	Peer-to-Peer	Mercury FDK SRC Carte
SGI RASC	NUMALink	none*	Master-Slave	various
HPTi HNPC	Ethernet	Linux	Master-Slave	HDL-based

Table 2.1: Comparison of HPRC Architectures

\* The SGI RASC must be connected to a cluster of GPPs.

HDL-based design and synthesis tools, or by using Annapolis’s CoreFire[3] module-based design suite.

Table 2.1 shows a comparison of the HPRC architectures discussed in this section. Of particular importance is the right-most column, which describes how each machine is programmed. Though there has been some effort towards standardization of interfaces and programming methods[43], currently the differences far outweigh the similarities.

## 2.3 Related Work

Developing an HPRC architecture is a process that contains many variables and encompasses a very large design space. Likewise, mapping applications to an HPRC machine is a difficult task. However, in the past few years the number of applications developed for these architectures has increased dramatically. This leads to two important results: first, some “best practices” have been discovered relating to the design of applications for HPRC machines; second, important information to guide the development of future HPRC architectures is now available. With this in mind,

in this section we survey recent applications and present some of these developments.

### 2.3.1 Backprojection

Backprojection itself is a well-studied algorithm. Most researchers focus on implementing backprojection for Computed Tomography (CT) medical imaging applications; backprojection for Synthetic Aperture Radar (SAR) on FPGAs is not well-represented in the literature.

The precursor to this work is that of Coric, Leiser, et al.[14] Backprojection for CT uses the “spotlight” mode of imaging, in which the sensing array is rotated around the target area. (Contrast this with the “stripmap” mode described in Section 2.1.) Coric’s implementation on an FPGA achieved a 20x speedup over a software-only version, using a 4-way parallel pipeline, clever memory caching, and appropriately-sized integer data words to reduce the required I/O bandwidth. Gac, Mancini, and Desvignes[20] use a similar technique on later-model FPGAs with embedded microprocessors (the Xilinx Virtex2Pro[60]). Their novel caching structure allows them to create a 2D backprojection kernel suitable for use in a 3D backprojection application, but achieves only 8x speedup.

CT backprojection has also been implemented on several other computing devices besides FPGAs. Indeed, new coprocessor devices such as graphics processing units (GPU)[22] and the Cell broadband engine[27] are becoming popular targets for application acceleration. Xue, Cheryuaka, and Tubbs[61] compare 2D and 3D backprojection implementations on GPUs and FPGAs. They show excellent results

(24x speedup on an FPGA in 2D, 44x and 71x speedup on a GPU for 2D and 3D, respectively) but do not include data transfer times in their performance results. This is a frequently-seen but significant error. Bockenbach et al.[6] implement 3D CT on GPUs, FPGAs, and the Cell, and show that the Cell has the potential for the best performance of the three. The variability of these results across researchers suggest that the application itself is not inherently suited for any one coprocessor in particular; rather, the performance gained comes from the skill with which the implementation is tuned to the platform in question.

Finally, Neri-Calderon et al.[41] implement CT on a DSP (digital signal processor) and list several other implementations; the interested reader is referred to their paper for a more exhaustive survey of backprojection implementations.

Of the implementations of backprojection for SAR, almost none have been designed for FPGAs. Soumekh and his collaborators have published on implementations on SAR in general and backprojection in particular[42], as well as Soumekh's reference book on the subject[51], but they do not examine the use of FPGAs for computation. Some recent work on backprojection for SAR on parallel hardware has come from Halmstad University in Sweden[24, 1]; their publications lay important groundwork but have not been implemented except in software and/or simulation.

The work presented in this thesis is based on our previous publications on the subject[11, 12, 13].

### 2.3.2 Other Applications

Backprojection is not the only application that has been mapped to HPRC platforms, though signal processing is traditionally a strength of RC and so large and complex signal processing applications like backprojection are common. With the emergence of HPRC, scientific applications are also seeing significant research effort. Among these are such applications as hyperspectral dimensionality reduction[18], equation solvers[9, 40], and small kernels such as matrix multiplication and FFTs[56, 50].

Buell et al.[9] implement a boolean equation solver on the SRC-6 using CARTE. They praise the CARTE programming environment, and give high marks to the SRC-6 for access to memory and available I/O throughput. While the physical aspects of the HPRC system have significant effects on the ultimate performance of applications, it is important not to lose sight of the amount of developer effort that goes into porting these applications.

Underwood and Hemmert[56], who look at linear algebra kernels, stress the importance of overlapping the transfer of data between GPP and FPGA with FPGA execution. Hiding I/O latency in this fashion is a critical aspect of high-performance application design. The authors also make the case for a redesign of the APIs used to control FPGA coprocessors, arguing that non-blocking API calls could significantly increase performance as well as enabling parallel execution of useful functions on the GPP.

Larger scientific applications that have seen significant effort in HPC research are

now being mapped to HPRC systems. One of the most popular application domains is Molecular Dynamics (MD). At the core, MD applications compute a series of pairwise interactions between the members of a large set of particles. Several different algorithms are available, but in general they all exhibit a large degree of parallelism.

Scrofano and Prasanna[47] achieve a modest 2-3x speedup of MD on the SRC-6 platform. They note two critical aspects of HPRC application design: partitioning of work between the GPP and the FPGA, and strength reduction of certain functions that are difficult to implement in hardware (in their case,  $e^x$ ). Alam et al.[2] use CARTE to map MD to the SRC-6, showing 3-4x speedup including data transfer. Smith, Vetter, and Liang[50] cite slightly better results on the SRC-6 (5-10x speedup), and also stress the role of data transfer in determining performance. Meredith, Alam, and Vetter[37] also achieve 5-10x speedup, performing MD on Cell and a GPU system.

The Floyd-Warshall shortest-path algorithm is examined by Bondhugula et al.[7]. The authors achieve 4-15x speedup on a Cray XD1, and explain the impact of data transfer times on the performance of their implementation. A previous paper that details their FPGA kernel implementation[8] uses a method of memory access to enable parallelism that is similar to that used in our backprojection implementation.

Of these applications, several common threads are visible. The transfer of data between GPP and FPGA can significantly impact performance. The ability to determine and control the memory access patterns of the FPGA and the on-board memories is critical as well. Finally, sacrificing the accuracy of the results in favor

of using lighter-weight operations that can be more easily implemented on an FPGA can be an effective way of increasing performance. Along with these aspects of application performance, the effort required by a developer to describe their HPRC program should be considered.

### 2.3.3 Best Practices and Future Development

The Cray XD1 has emerged as a popular target for HPRC research, owing in part to its availability to researchers and the many different programming tools that target it. Tripp et al.[54] present a case study of application development for the XD1 and go into great detail about the design process, the application attributes that influence final performance, and ideas about how to balance them. Using a traffic simulation application with a Cellular Automata (CA) algorithm, they call out a number of important aspects that must be considered to create high-performance applications. These include: partitioning the work between the GPP and the FPGA, determining how much parallelism the FPGA design should exploit, optimizing data transfer between the compute nodes, and exploiting the ability of the GPP and FPGA to perform separate tasks in parallel.

Gokhale et al.[21] add to these conclusions by taking a step back and examining the problem of application mapping at a higher level. In order to get the highest performance from an HPRC application, they argue, developers must avoid the pitfall of starting from a software implementation and mapping portions of it straight to hardware. Instead, the overall architecture of the solution must be re-thought, taking

into account the capabilities of both the GPP and the FPGA and designing an application that can take advantage of both. Specifically, choosing an appropriate number representation format (a problem that software developers rarely need to worry about), proper partitioning of the work, and close attention paid to data transfer are all key to high-performance applications.

Herbordt et al.[25] also survey the state-of-the-art techniques for high-performance application development on reconfigurable computing devices. Though their list is not targeted specifically at HPRC, their methods are applicable to that field as well. Similarly to the previous two groups, Herbordt suggests that application designers consider the strengths and limitations of the hardware to which the application is being targeted. Reduction of computation strength, appropriate choice of number format, use of special-purpose hardware blocks contained within the FPGA, and matching of data input rate to processing rate are key on this list.

Rather than developing whole applications for HPRC machines, some researchers have taken the approach of developing a library of FPGA modules[39, 49, 57, 62] that implement certain kernels, or small functions, that are applicable to a number of different applications. It is hoped that by replacing parts of a software program with accelerated versions that run on FPGAs, the overall runtime of the software program can be decreased. While these kernels tend to outperform their software counterparts, the overall effect on an application's runtime is generally small, on the order of 2-3x speedup[21]. However, libraries (especially when used in conjunction

with a high-level language such as CARTE[52] or Mitrion-C[38]) generally provide a reduction in developer effort, leading to an improvement in application design time.

Finally, Sass et al.[46] have taken a long look into the future and attempted to identify the components that would go into a “Petascale” HPRC system. While the authors above have determined what is needed in order to make today’s applications run faster, Sass examines such system design aspects as packaging, power, cooling, required physical space, and storage. In addition, he argues the need for an increased focus on inter-node communication methods and greatly improved memory bandwidth in order to meet the higher demands for data that come along with increased performance.

## 2.4 Summary

In this chapter we introduced the backprojection algorithm and described the mathematical basis for the algorithm that we have implemented. A discussion of the current state of the art in HPRC system design reveals the features that either encourage or discourage high-performance application design. Finally, a review of recent HPRC application research and the emerging best practices provides context for our implementation.

We used several of these “best practices” in the design of our backprojection implementation. In particular, changing the format of the input data and reducing the strength of difficult arithmetic operations contribute greatly to a high-performance design. When considered in conjunction with the available memory bandwidth, an

appropriate level of parallel execution can be chosen. In the next chapter we describe our hardware implementation of backprojection for SAR and go into more detail about these design descisions.

# Chapter 3

## Experimental Design

In this chapter we describe an implementation of the backprojection image formation algorithm on a high-performance reconfigurable computer. It has been designed to fit into the Swathbuckler system, providing high-speed image formation services and supporting output data distribution via a publish/subscribe[19] methodology. Section 3.1 describes the Swathbuckler system that forms the basis of this project. Section 3.2 explores the inherent parallelism in backprojection and describes the high-level design decisions that steered the implementation. Section 3.3 describes the hardware implementation itself, and Section 3.4 discusses the supporting software program that runs on the GPP.

### 3.1 System Background

In the previous chapter we described the HHPC system and the Swathbuckler project. In this section we will explore more deeply the aspects of those systems that are relevant to our experimental design.

### 3.1.1 Swathbuckler Project

The Swathbuckler project[30, 45, 55] is an implementation of synthetic aperture radar created by a joint program between the American, British, Canadian, and Australian defense research project agencies. It encompasses the entire SAR process including the aircraft and radar dish, signal capture and analog-to-digital conversion, filtering, and image formation hardware and software.

As currently implemented, radar data is recorded, sampled digitally, and transferred to the HPTi HHPC (see Section 2.2.1) to be processed. Software on the HHPC runs a Fourier-based image formation algorithm, resulting in JPEG images of the target area. Output images are then disseminated via a publish/subscribe mechanism. Figure 3.1 shows a block diagram of the Swathbuckler system.

Our problem as posed was to increase the processing capabilities of the HHPC by increasing the performance of the portions of the application seen on the right-hand side of Figure 3.1. Given that a significant amount of work had gone into

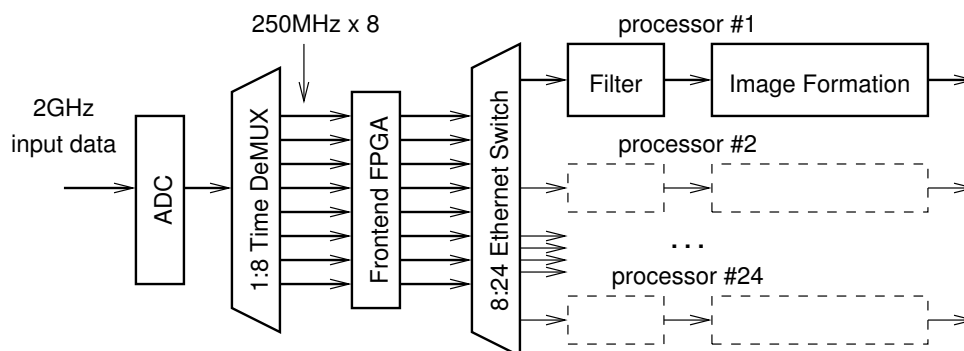


Figure 3.1: Block diagram of Swathbuckler system  
Adapted from [45]

tuning the performance of the software implementation of the filtering process[45], it remained for us to improve the speed at which images could be formed. According to the project specification, the input data is streamed into the microprocessor's main memory. In order to perform image formation on the FPGA, it is then necessary to copy data from the host to the FPGA. Likewise, the output image must be copied from the FPGA's memory to the host memory so that it can be made accessible to the publish/subscribe software. These data transfer times must be included in our performance measurements (see Chapter 4).

### 3.1.2 HHPC Features

Several features of the Annapolis WildStar II FPGA boards are directly relevant to the design of our backprojection implementation. These features were introduced in Section 2.2.1, shown in Figure 2.2, and will be described in more detail here. In particular, the host-to-FPGA interface, the on-board memory bandwidth, and the available features of the FPGA itself limit or guide our design decisions.

Communication between the host GPP and the WildStar II board travels over a PCI bus. The HHPC provides a PCI bus that runs at 66MHz with 64-bit data words. The WildStar II on-board PCI interface translates this into a 32-bit interface running at 133MHz. By implementing the DMA data transfer mode to communicate between the GPP and the FPGA, the on-board PCI interface performs this translation invisibly and without significant loss of performance. A 133MHz clock is also a good and achievable clock rate for FPGA hardware, so most of the hardware design

can be run directly off the PCI interface clock. This simplifies the design since there are fewer clock domains to cross (see Section 3.3.1).

The WildStar II board has six on-board SRAM memories (1MB each) and one SDRAM memory (64MB). It is beneficial to be able to read one piece of data and write one piece of data in the same clock cycle, so we prefer to use multiple SRAMs instead of the single larger SDRAM. The SRAMs run at 50MHz and feature a 32-bit dataword (plus four parity bits), but they use a DDR interface. The Annapolis controller for the SRAM translates this into a 50MHz 72-bit interface. Both of these features are separately important: we will need to cross from the 50MHz memory clock domain to the 133MHz PCI clock domain, and we will need to choose the size of our data such that it can be packed into a 72-bit memory word (see Section 3.2.4).

Finally, the Virtex2 6000 FPGA on the Wildstar II has some useful features that we use to our advantage. A large amount of on-chip memory is available in the form of BlockRAMs, which are configurable in width and depth but can hold at most 2KB of data. 144 of these dual-ported memories are available, each of which can be accessed independently. This makes BlockRAMs a good candidate for storing and accessing input projection data (see Sections 3.2.4 and 3.3.3). BlockRAMs can also be configured as FIFOs, and due to their dual-ported nature, can be used to cross clock domains.

## 3.2 Algorithm Analysis

In this section we dissect the backprojection algorithm with an eye towards implementing it on an HPRC machine. As discussed in Section 2.3.3, there are many factors that need to be taken into account when designing an HPRC application. First and foremost, an application that does not have a high degree of parallelism is generally not a good candidate. We must then decide how to divide the problem along the available lines of parallelism, to determine what part of the application will be executed on each available processor. This includes GPP/FPGA assignment as well as dividing the problem across the multiple nodes of the cluster. For the portions of the application run on the FPGAs, data arrays must be distributed among the accessible memories. Next, we look at some factors to improve the performance of the hardware implementation, namely, data formats and computation strength reduction. We conclude by examining the parameters of the data collection process that affect the computation.

### 3.2.1 Parallelism Analysis

In any reconfigurable application design, performance gains due to implementation in hardware inevitably come from the ability of reconfigurable hardware (and, indeed, hardware in general) to perform multiple operations at once. Extracting the *parallelism* – or, determining where operations may be performed in parallel – from an application is thus critical to a high-performance implementation.

In Section 2.1, Equation (2.1) showed the backprojection operation in terms of projection data  $p(t, u)$  and an output image  $f(x, y)$ . That equation may be interpreted to say that for a particular pixel  $f(\hat{x}, \hat{y})$ , the final value can be found from a summation of contributions from the set of all projections  $p(t, u)$  whose corresponding radar pulse covered that ground location, with the value of  $t$  for a given  $u$  determined by the mapping function  $i(x, y, u)$  according to Equation (2.2). There is a large degree of parallelism inherent in this interpretation:

1. The contribution from projection  $p(\hat{u})$  to pixel  $f(\hat{x}, \hat{y})$  is not dependent on the contributions from all other projections  $p(u)$ ,  $u \neq \hat{u}$  to that same pixel  $f(\hat{x}, \hat{y})$ .
2. The contribution from projection  $p(\hat{u})$  to pixel  $f(\hat{x}, \hat{y})$  is not dependent on the contribution from  $p(\hat{u})$  to all other pixels  $f(x, y)$ ,  $x \neq \hat{x}$ ,  $y \neq \hat{y}$ .
3. The final value of a pixel is not dependent on the value of any other pixel in the target image.

It can be said, therefore, that backprojection is an “embarrassingly parallel” application, which is to say that it lacks any data dependencies. Without data dependencies, the opportunity for parallelism is vast and it is simply a matter of choosing the dimensions along which to divide the computation that best match the system on which the algorithm will be implemented.

### 3.2.2 Dividing The Problem

There are two ways in which parallel applications are generally divided across the nodes of a cluster.

1. Split the *data*. In this case, each node performs the same computation as every other node, but on a subset of data. There may be several different ways that the data can be divided.
2. Split the *computation*. In this case, each node performs a portion of the computation on the entire dataset. Intermediate sets of data flow from one node to the next. This method is also known as task-parallel or systolic computing.

While certain supercomputer networks may make the task-parallel model attractive, our work with the HHPC indicates that its architecture is more suited to the data-parallel mode. Since inter-node communication is accomplished over a many-to-many network (Ethernet or Myrinet), passing data from one node to the next as implied by the task-parallel model will potentially hurt performance. A task-parallel design also implies that a new FPGA design must be created for each FPGA node in the system, greatly increasing design and verification time. Finally, the number of tasks available in this application is relatively small and would not occupy the number of nodes that are available to us.

Given that we will create a data-parallel design, there are several axes along which we considered splitting the data. One method involves dividing the input projection

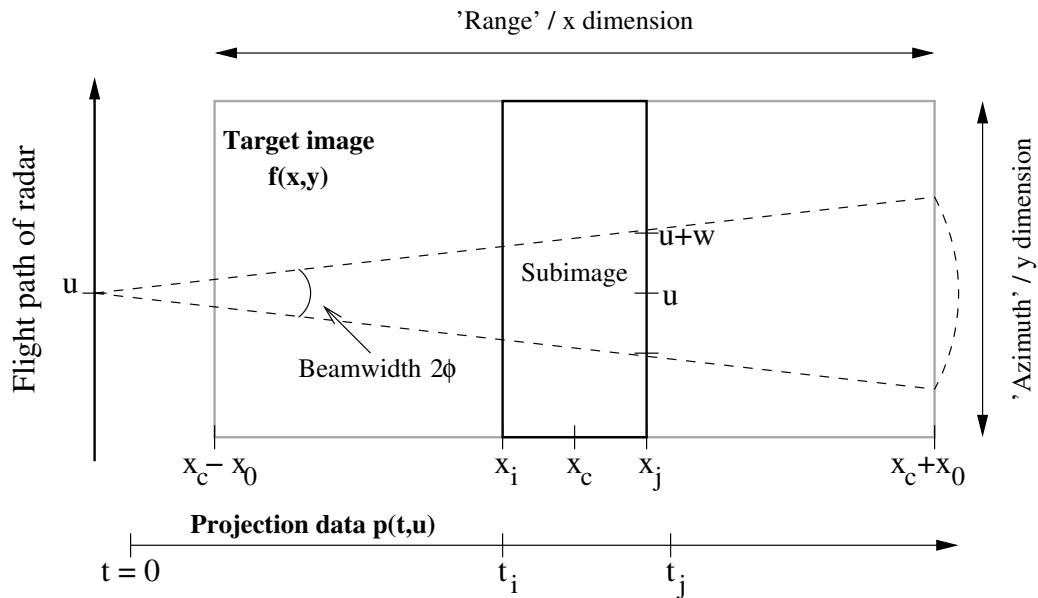


Figure 3.2: Division of target image across multiple nodes

data  $p(t, u)$  among the nodes along the  $u$  dimension. Each node would hold a portion of the projections  $p(t, [u_i, u_j])$  and calculate that portion's contribution to the final image. However, this implies that each node must hold a copy of the entire target image in memory, and furthermore, that all of the partial target images would need to be added together after processing before the final image could be created. This extra processing step would also require a large amount of data to pass between nodes. In addition, the size of the final image would be limited to that which would fit on a single FPGA board.

Rather than divide the input data, the preferred method divides the output image  $f(x, y)$  into pieces along the range ( $x$ ) axis (see Figure 3.2). In theory, this requires that every projection be sent to each node; however, since only a portion of each projection will affect the slice of the final image being computed on a single node,

only that portion must be sent to that node. Thus, the amount of input data being sent to each node can be reduced to  $p([t_i, t_j], u)$ . We refer to the portion of the final target image being computed on a single node,  $f([x_i, x_j], y)$ , as a “subimage”.

Figure 3.2 shows that  $t_j$  is slightly beyond the time index that corresponds to  $x_j$ . This is due to the width of the cone-shaped radar beam. The dotted line in the figure shows a single radar pulse taken at slow-time index  $y = u$ . The minimum distance to any part of the subimage is at the point  $(x_i, u)$ , which corresponds to fast-time index  $t_i$  in the projection data. The maximum distance to any part of the subimage, however, is along the outer edge of the cone to the point  $(x_j, u \pm w)$  where  $w$  is a factor calculated from the beamwidth angle of the radar and  $x_j$ . Thus the fast-time index  $t_j$  is calculated relative to  $x_j$  and  $w$  instead of simply  $x_j$ . This also implies that the  $[t_i, t_j]$  range for two adjacent nodes will overlap somewhat, or (equivalently) that some projection data will be sent to more than one node.

Since the final value of a pixel does not depend on the values of the pixels surrounding it, each FPGA need hold only the subimage that it is responsible for computing. That portion is not affected by the results on any other FPGA, which means that the post-processing accumulation stage can be avoided. If a larger target image is desired, subimages can be “stitched” together simply by concatenation.

In contrast to the method where input data are divided along the  $u$  dimension, the size of the final target image is not restricted by the amount of memory on a single node, and furthermore, larger images can be processed by adding nodes to the

cluster. This is commonly referred to as *coarse-grained parallelism*, since the problem has been divided into large-scale independent units. Coarse-grained parallelism is directly related to the performance gains that are achieved by adapting the program from a single-node computer to a multi-node cluster.

### 3.2.3 Memory Allocation

The memory devices used to store the input and output data may now be determined. On the Wildstar II board there are three options: an on-board DRAM, six on-board SRAMs, and a variable number of BlockRAMs which reside inside the FPGA and can be instantiated as needed. The on-board DRAM has the highest capacity (64MB) but is the most difficult to use and only has one read/write port. BlockRAMs are the most flexible (two read/write ports and a flexible geometry) and simple to use, but have a small (2KB) capacity. We need to store two large arrays of information: the target image  $f(x, y)$  and the input projection data  $p(t, u)$ .

For the target image, we would like to be able to both read and write one target pixel per cycle. It is also important that the size of the target image stored on one node be as large as possible, so memories with larger capacity are better. Thus we will use multiple on-board SRAMs to store the target image. By implementing a two-memory storage system, we can provide two logical ports into the target image array. During any given processing step, one SRAM acts as the source for target pixels, and the other acts as the destination for the newly computed pixel values. When the next set of projections is sent to the FPGA, the roles of the two SRAMs

are reversed.

For the projection data, we would like to have many small memories that can each feed one of the projection adder units. BlockRAMs allow us to instantiate multiple small memories in which to hold the projection data; each memory has two available ports, meaning that two adders can be supported in parallel. Since there are six SRAMs, we can create two pipelines of adders (see Section 3.3.3). Each pipeline has one read/write pair of SRAMs (four used in total) and a set of BlockRAMs feeding the adders (limited by the number of available BlockRAMs on the FPGA).

### 3.2.4 Data Formats

Backprojection is generally accomplished in software using a complex (i.e., real and imaginary parts) floating-point format. However, since the result of this application is an image which requires only values from 0 to 255 (i.e. 8-bit integers), the loss of precision inherent in transforming the data to a fixed-point/integer format is negligible. In addition, using an integer data format allows for much simpler functional units (see next section).

Given an integer data format, it remains to determine how wide the various data words should be. We base our decision on the word width of the memories. The SRAM interface provides 72 bits of data per cycle, comprised of two physical 32-bit datawords plus four bits of parity each. The BlockRAMs are configurable, but generally can provide power-of-two sized datawords.

Since the application of backprojection is in essence an accumulation operator,

it makes sense for the output data (target image pixels) to be wider than the input data (projection samples). This reduces the likelihood of overflow error in the accumulation. Matching the width of the data ports into their respective memories, we will therefore use 36-bit complex integers (18-bit real and 18-bit imaginary) for the target image, and 32-bit complex integers for the projection data.

After backprojection, a complex magnitude operator is needed to reduce the 36-bit complex integers down to a single 18-bit real integer. This operator is implemented in hardware, but the process of scaling data from 18-bit integer to 8-bit image is left to the software running on the GPP.

### 3.2.5 Strength Reduction

The computation to be performed on each node consists of three parts. The summation from Equation 2.1 and the distance calculation from Equation 2.2 represent the backprojection work to be done. The complex magnitude operation is similar to the distance calculation.

While adders are simple to replicate in large numbers, the hardware required to perform multiplication and square root is more costly. If we were using floating-point data formats, the number of functional units that could be instantiated would be very small, reducing the amount of parallelism that we can exploit. With integer data types, however, these units are relatively small, fast, and easily pipelined. This allows us to maintain a high clock rate and one-result-per-cycle throughput.

### 3.2.6 Data Collection Parameters

The conditions under which the projection data are collected affect certain aspects of the backprojection computation. In particular, the spacing between samples in the  $p(t, u)$  array and the spacing between pixels in the  $f(x, y)$  array imply constant factors that must be accounted for during the distance-to-time index calculation (see Section 3.3.3).

For the input data,  $\Delta u$  indicates the distance (in meters) between samples in the azimuth dimension. This is equivalent to the distance the plane travels between each outgoing pulse of the radar. Often, due to imperfect flight paths, this value is not actually accurate. The data filtering that occurs prior to backprojection image formation is responsible for correcting for inaccuracies due to the actual flight path, so that a regular spacing can be assumed.

As the reflected radar data is observed by the radar receiver, it is sampled at a particular frequency  $\omega$ . That frequency translates to a range distance  $\Delta t$  between samples equal to  $c/2\omega$ , where  $c$  is the speed of light. The additional factor of  $1/2$  accounts for the fact that the radar pulse travels the intervening distance, is reflected, and travels the same distance back. Owing to the fact that the airplane is not flying at ground level, there is an additional angle of elevation that is included to determine a more accurate value for  $\Delta t$ .

For the target image (output data),  $\Delta x$  and  $\Delta y$  simply correspond to the real distance between pixels in the range and azimuth dimensions, accordingly. In general,

$\Delta x$  and  $\Delta y$  are not necessarily related to  $\Delta u$  and  $\Delta t$  and can be chosen at will. In practice, setting  $\Delta y = \Delta u$  makes the algorithm's computation more regular (and thus more easily parallelizable). Likewise, setting  $\Delta x = \Delta t$  reduces the need for interpolation between samples in the  $t$  dimension since most samples will line up with pixels in the range dimension. Finally, setting  $\Delta x = \Delta y$  provides for square pixels and an easier-to-read aspect ratio in the output image.

The final important parameter is the minimum range from the radar to the target image, known as  $R_{min}$ . This is related to the  $t_i$  parameter as discussed in Section 3.2.2, and is used by the software to determine what portion of the projection data is applicable to a particular node.

The use of these parameters is further discussed in Section 3.4.1.

### 3.3 FPGA Design

This section details the design of the hardware that runs on the FPGA boards which are attached to the host GPPs. This hardware 'program' runs the backprojection algorithm and computes the unnormalized output image. A block diagram of the design can be seen in Figure 3.3; this figure will be referred to throughout this section. Throughout this section, references to blocks in Figure 3.3 will be printed in monospace.

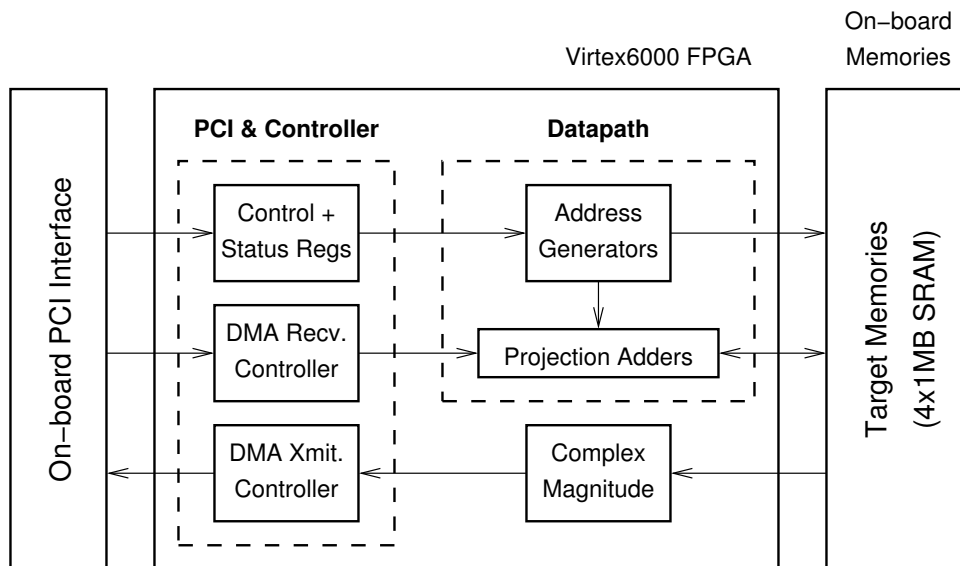


Figure 3.3: Block diagram of backprojection hardware unit

### 3.3.1 Clock Domains

In general, using multiple clock domains in a design adds complexity and makes verification significantly more difficult. However, the design of the Annapolis Wildstar II board provides for one fixed-rate clock on the PCI interface, and a separate fixed-rate clock on the SRAM memories. Since we intend to use both of these elements in our design, we must account for two clock domains.

To simplify the problem, we run the bulk of our design at the PCI clock rate (133MHz). Since Annapolis' VHDL modules refer to the PCI interface as the “LAD bus”, we will call this the **L-clock** domain. Every block in Figure 3.3, with the exception of the SRAMs themselves and their associated **Address Generators**, is run from the L-clock.

The SRAMs are run from the memory clock, or **M-clock**, which is constrained to

run at 50MHz. Between the `Target Memories` and the `Projection Adders`, there is some interface logic and a FIFO. This is not shown on Figure 3.3 but it exists to cross the M-clock/L-clock domain boundary.

BlockRAM-based FIFOs, available as modules in the Xilinx CORE Generator[58] library, can be used to cross clock domains. Since each of the ports on the dual-ported BlockRAMs is individually clocked, the read and write can happen in different clock domains. Control signals are automatically synchronized to the appropriate clock, i.e. the “full” signal is synchronized to the write clock and the “empty” signal to the read clock. Consistently using FIFOs whenever clock domains must be crossed provides a simple and well-tested solution.

### 3.3.2 Control Registers and DMA Input

The Annapolis API allows for communication between the host PC and the FPGA with two methods: “Programmed” or memory-mapped I/O (PIO), which is best for reading and writing small data sets like control registers, and Direct Memory Access (DMA), which is best for transferring large blocks of data.

`Control and Status Registers` (CSRs) are read and written via PIO. These registers allow the host software to coordinate with the hardware program. One block of CSRs is used for certain hardware initialization functions, such as requesting that the contents of the SRAMs be zeroed and checking the condition of the on-chip clocks. Since the API demands that the FPGA master all DMA transactions, another block of CSRs are set aside for DMA transaction handshaking and control. The flight

parameters in Section 3.2.6 are also set via CSRs. The remaining CSRs are used to control the processing of a batch of projection data.

The host software is responsible for organizing the input projection data so that it can be copied to the FPGA board via DMA. A DMA buffer is allocated on the host side using API calls. Before each processing step, the host determines which portion of the  $p(t, u)$  array should be copied into the DMA buffer. Once this copy is complete the FPGA is signaled using the DMA control CSRs.

The hardware program, in turn, contains a DMA Receive Controller (DRC) that requests the data in the DMA buffer in host memory. As that data is received by the FPGA, it is distributed to the projection BlockRAMs by the DRC. Because the Projection Adders (and the BlockRAMs contained within) are also run in the L-clock domain, this is a straightforward task.

### 3.3.3 Datapath

The main datapath of the backprojection hardware is shown in Figure 3.4. It consists of five parts: the Target Memory SRAMs that hold the target image, the Distance-to-Time Index Calculator (DIC), the Projection Data BlockRAMs, the Adders to perform the accumulation operation, and the Address Generators that drive all of the memories. These devices all operate in a synchronized fashion, though there are FIFOs in several places to temporally decouple the producers and consumers of data. (FIFOs are indicated in Figure 3.4 with segmented rectangles.)

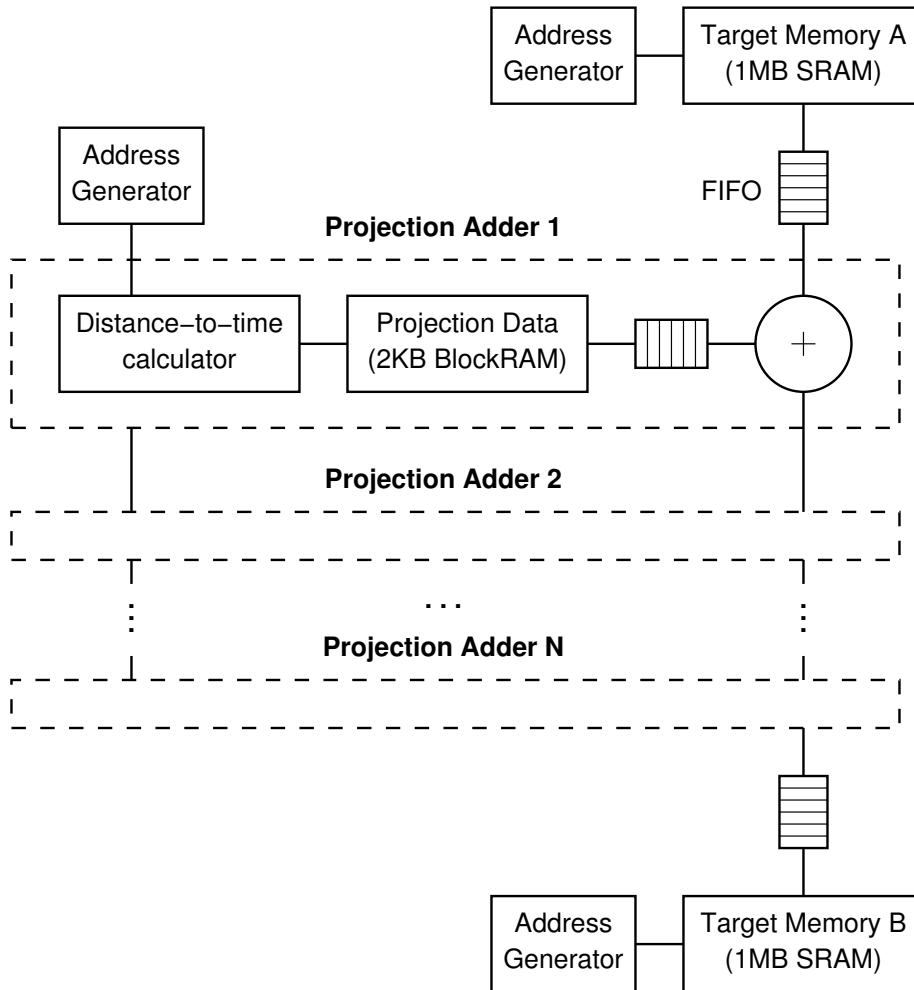


Figure 3.4: Block diagram of hardware datapath

## Address Generators

There are three data arrays that must be managed in this design: the input target data, the output target data, and the projection data. The pixel indices for the two target data arrays (**Target Memory A** and **Target Memory B** in Figure 3.4) are managed directly by separate address generators. The address generator for the **Projection Data BlockRAMs** also produces pixel indices; the DIC converts the pixel index into a fast-time index that is used to address the BlockRAMs.

Because a single read/write operation to the SRAMs produces/consumes two pixel values, the address generators for the SRAMs run for half as many cycles as the address generator for the BlockRAMs. However, address generators run in the clock domain relevant to the memory that they are addressing, so  $n/2$  SRAM addresses takes slightly longer to generate at 50MHz than  $n$  BlockRAM addresses at 133MHz.

Because of the use of FIFOs between the memories and the adders, the address generators for the **Target Memory A** and the **Projection Data BlockRAMs** can be run freely. FIFO control signals ensure that an address generator is paused in time to prevent it from overflowing the FIFO. The address generator for **Target Memory B** is incremented whenever data are available from the output FIFO.

## Target Memory SRAMs

Owing to the 1MB size of the SRAMs in which we store the target image data, we are able to save  $2^{19}$  pixels. We choose to arrange this into a target image that is 1024 pixels in the azimuth dimension and 512 in the range dimension. Using power-of-

two dimensions allows us to maximize our use of the SRAM, and keeping the range dimension small allows us to reduce the amount of projection data that must be transferred.

### Distance-To-Time Index Calculator

The Distance-To-Time Index Calculator (DIC) implements Equation (2.2), reproduced below, which is comprised of two parts. At first glance, each of these parts involves computation that requires large amount of hardware and/or time to calculate. However, a few simplifying assumptions make this problem easier and reduce the amount of needed hardware.

$$i(x, y, u) = \chi(y - x \tan \phi, y + x \tan \phi) \cdot \frac{\sqrt{x^2 + (y - u)^2}}{c} \quad (2.2)$$

Rather than implement a tangent function in hardware, we rely on the fact that the beamwidth of the radar is a constant. The host code is required to perform the  $\tan \phi$  function and upload the result to a CSR, which is then used to calculate  $\chi(a, b)$ . This value is used to both coarsely narrow the range of pixels which are examined for each processing step, and on a fine-grained level to determine whether or not a particular pixel is affected by the current projection (see Figure 3.2).

The right-hand side of Equation (2.2) is a distance function ( $\sqrt{x^2 + y^2}$ ) and a division. Because we are using integer datatypes, the square root function can be executed using an iterative shift-and-subtract algorithm. In hardware, this algorithm is implemented with a pipeline of subtractors. Two multiplication units handle the  $x^2$  and  $y^2$  functions. Some additional adders and subtractors are necessary to properly

align the input data to the output data according to the data collection parameters discussed in Section 3.2.6.

The distance function and computation of  $\chi()$  occur in parallel. If the  $\chi()$  function determines that the pixel is outside the affected range, the adder's input is forced to zero.

### Projection Data BlockRAMs

The output of the DIC is a fast-time index into the  $p(t, u)$  array. Each **Projection Data BlockRAM** holds the data for a particular value of  $u$ . The fast-time index  $t$  is applied to retrieve a single value of  $p(t, u)$  that corresponds to the pixel that was input by the address generator. This value is stored in a FIFO, to be synchronized with the output of the **Target Memory A FIFO**.

**Projection Data** memories are configured to hold 2k data words by default, which should be sufficient for a 1k range pixel image. This number is a compile-time parameter in the VHDL source and can be changed. The resource constraint is the number of available BlockRAMs.

### Projection Adder

As the FIFOs from the **Projection Data** memories and the **Target Memory** are filled, the **Projection Adder** reads data words from both FIFOs, adds them together, and passes them to the next stage in the pipeline (see Figure 3.4). Control signals flow from Adder 1 to Adder N to ensure that **Projection Data** FIFOs are read at the right time as intermediate results flow down the adder pipeline.

The design is configured with eight adder stages, meaning eight projections can be processed in one step. This number is a compile-time parameter in the VHDL source and can be changed. The resource constraint is a combination of the number of available BlockRAMs (because the `Projection Data` BlockRAMs and FIFO are duplicated) and the amount of available logic (to implement the DIC).

The number of adder stages implemented directly impacts the performance of our application. By computing the contribution of multiple projections in parallel, we exploit the *fine-grained parallelism* inherent in the backprojection algorithm. Fine-grained parallelism is directly related to the performance gains achieved by implementing the program in hardware, where many small execution units can be implemented that all run at the same time on different pieces of data.

### 3.3.4 Magnitude Computation and DMA Output

When all projections have been computed, the final target image data resides in one of the `Target Memory` SRAMs. (A single CSR bit keeps track of which SRAM contains the latest data.) The host software uses the CSRs to indicate the parameters of the DMA transaction and request a transfer of the final target data. The FPGA then begins reading the data from the correct `Target Memory` SRAM and transmits them via DMA to the host's memory.

However, the data stored in the SRAMs are complex numbers, that is, they are comprised of real and imaginary parts. As a first step to forming an image with values 0-255, the complex numbers must be converted to real. The `Complex`

Magnitude operator performs this function with a distance calculation ( $\sqrt{re^2 + im^2}$ ).

We instantiate another series of multipliers, adders, and subtractors (for the integer square root) to perform this operation.

As data are read out of the SRAM, the complex magnitude operation converts them to real numbers. The real numbers are then passed to the DMA Transmit Controller (DXC) and sent out to the PCI bus.

## 3.4 Software Design

The hardware program can not run on its own. It requires control from a GPP to which it is attached. In this section we explain the purpose and responsibilities of the host program and how it interacts with the FPGA. Further, we explain the adaptations necessary to run the backprojection program on a multi-node supercomputer.

### 3.4.1 Single-node software

The host program begins by initializing the API that allows for communication between the host and FPGA. The FPGA program is loaded, clocks are initialized, and the hardware is reset to put it into a known state before processing.

The backprojection program must be told the flight parameters from Section 3.2.6. In particular, the values of  $\Delta_x$ ,  $\Delta_y$ , and  $R_{min}$  (the minimum range) are required. However, in order to avoid the use of fractional numbers, all of these parameters are normalized such that  $\Delta_t = 1$ . This allows the hardware computation to be in terms of fast-time indices in the  $t$  domain instead of ground distances. However,

requiring these parameters to be integer ratios is somewhat restrictive.

Before processing can begin on the FPGA, the host program must receive the radar data, organize it, and prepare it for transmission to the FPGA. First, the input data is in complex floating-point format and must be translated to integers. Second, the host software must determine what range of  $t$  is relevant to the subimage being calculated (see Section 3.2.2). Note that because  $t_j > x_j$ , there is some overlap between subimages.

The Projection Data BlockRAMs contain  $2^{R+9}$  samples in the  $t$  domain.  $R$  is a VHDL compile-time parameter, but is also available to be read via CSR from the hardware. It defaults to 2, meaning  $2^{11} = 2048$  samples are stored for each projection. Likewise, the number of projection adders,  $N$ , is also available. The host program allocates a data buffer that is large enough to hold  $N \cdot 2^{R+9}$  samples for each projection adder. A loop then begins, where the appropriate amount of projection data is copied into that buffer, then transmitted to the FPGA and processed. After the FPGA processes the data, the roles of the Target Memory SRAMs are switched, and another iteration of the loop begins. When all iterations are complete (and all projection data have been processed), the host program requests that the target data be transmitted from the FPGA into the GPP's memory. The data are scaled, rearranged into an image buffer, and an image file is produced using a library call to the GTK+ library[23].

### 3.4.2 Multi-node parallel software

For the Swathbuckler system, projection data is streamed into the nodes as it is received by the radar and “front-end” processing system; refer to [45] for more detail. According to Figure 3.2, each FPGA node of our backprojection implementation processes a subimage that corresponds to a portion of the projection data in the  $t$  domain for each particular value of  $u$ . Thus, incoming projection data, after filtering, must be divided up and distributed to the individual nodes. (The distribution of data to the particular nodes is outside the scope of this thesis.) The data is then processed according to the information in the previous section.

When all the projection data has been received, the subimages are simply held in the GPP’s memory. Subimages are distributed to consumers via a publish/subscribe mechanism, so there is no need to assemble all the subimages into a larger image.

Since we are not able to integrate directly into the Swathbuckler system, our experiment begins by reading a complete post-filtering input data set from the shared disk array attached to the HHPC cluster. Using the MPI library to start and coordinate processing across multiple nodes, each node reads the portion of the disk file that it is responsible for processing. Projections are processed as quickly as possible and the target subimages are formed. Because of the embarrassingly parallel nature of backprojection, no data transfer is necessary between the nodes. This should result in nearly-linear speedup in the number of nodes; i.e. producing 64 subimages on 8 nodes should be roughly twice as fast as processing the same data on 4 nodes.

We will discuss the experimental setup, including exactly what processing steps are measured, in Chapter 4.

## 3.5 Summary

This chapter describes the hardware implementation of backprojection that we have developed. Using the principles discussed in Chapter 2 and a working knowledge of the HPRC system on which the application is run, we provided details of the design decisions that went into the development of the implementation and the impact that we expect them to have on the performance of the application. Furthermore, a brief outline of the software program that controls the application is included.

In the next chapter we will describe the experiments that were run. Though accuracy is obviously one critical aspect of our design, high performance is an important concern as well. We will show the performance benefits of running backprojection in hardware and explain how the design decisions from this chapter affect that performance.

# Chapter 4

## Experimental Results

After completing the design, we conducted a series of experiments to determine the performance and accuracy of the hardware. When run on a single node, a detailed profile of the execution time of both the software and hardware programs can be determined, and the effects of reconfigurable hardware design techniques can be studied. Running the same program on multiple nodes shows how well the application scales to take advantage of the processing power available on HPC clusters. In this chapter we describe the experiments and analyze the collected results.

### 4.1 Experimental Setup

Our experiments consist of running programs on the HHPC and measuring the run time of individual components as well as the overall execution time. There are two programs: one which forms images by running backprojection on the GPP (the “software” program), and one which runs it in hardware on an FPGA (the “hardware” program).

We are concerned with two factors: speed of execution, and accuracy of solution.

We will consider the execution time of the backprojection operation by itself, but also the execution time of the entire program including setup and teardown operations. In the context of the Swathbuckler project, the primary concern is the speed of backprojection. However, reconfigurable computing application design is often concerned with full application run time, so we will also examine this aspect.

In terms of accuracy, the software program computes its results in floating point while the hardware uses integer arithmetic. We will examine the differences between the images produced by these two programs in order to establish the error introduced by the data format conversion.

The ability of our programs to scale across multiple nodes is an additional metric of performance. We measure the effectiveness of exploiting coarse-grained parallelism by comparing the relative performance of both programs when run on one node and when run on many nodes.

### 4.1.1 Program Structure

Each of the experiments was conducted on the same platform. Nodes on the HHPCC run the Linux operating system, RedHat release 7.3, using kernel version 2.4.20. Both the software and hardware programs are written in C. The hardware program uses an API (version 199936) that is provided by Annapolis to interface to the WildStar II boards. Both the software and hardware programs use the GTK+[23] library version 1.2.0 to create the target images. For the multi-node experiments, version 1.2.4 of the MPICH[5] implementation of MPI is used to handle inter-node startup,

communication, and teardown. These libraries have been pre-installed on the HHPC.

FPGA hardware was written in VHDL and synthesized using version 8.2 of Synplify Pro. Hardware place and route used the Xilinx ISE 8.1i suite of CAD tools.

Timing data was collected using internal timing routines inserted into the C code. Measurements of these routines indicate an overhead of less than  $100\mu\text{s}$ , so results are presented as accurate to the millisecond. Unless noted otherwise, applications were run five times and an average (arithmetic mean) was taken to arrive at the presented data.

We determine accuracy both qualitatively (i.e. by comparing the images with the human eye) and quantitatively by computing the difference in pixel values between the two images.

### 4.1.2 Test Data

Four sets of data were used to test our programs. The data sets were produced using a MATLAB simulation of SAR taken from Soumekh's book on SAR[51]. This MATLAB script allows the parameters of the data collection process to be configured (see Section 3.2.6). When run, it generates the projection data that would be captured by a SAR system imaging that area. A separate C program takes the MATLAB output and converts it to an optimized file that can be read by the backprojection programs.

Each data set contains four point-source (i.e.  $1\times 1$  pixel in size) targets that are distributed randomly through the imaged area. The imaged area for each set is of a

Component	Software	Hardware	Ratio
Initialization	21 ms	701 ms	1:38.6
<i>Backprojection</i>	<i>76.4 s</i>	<i>351 ms</i>	<i>217:1</i>
Complex Magnitude	73 ms	15 ms	4.9:1
Form Image (software)	39 ms	340 ms	1:8.7
Total	76.5 s	1.41 s	54.4:1

Table 4.1: Single-node experimental performance

similar size, but situated at a different distance away from the radar. Targets are also assigned a random reflectivity value that indicates how strongly they reflect radar signals. More information on the data sets is available in Appendix A.

## 4.2 Results and Analysis

In general, owing to the high degree of parallelism inherent in the backprojection algorithm, we expect a considerable performance benefit from implementation in hardware even on a single node. For the multi-node program, the lack of need to transfer data between the nodes implies that the performance should scale in a linear relation to the number of nodes.

### 4.2.1 Single Node Performance

The first experiment involves running backprojection on a single node. This allows us to examine the performance improvement due to *fine-grained parallelism*, that is, the speedup that can be gained by implementing the algorithm in hardware. For this experiment, we ran the hardware and software programs on all four of the data sets. Table 4.1 shows the timing breakdown of data set #4; Table 4.2 shows the overall results for all four data sets.

In Table 4.1, **Software** and **Hardware** refer to the run time of a particular component; **Ratio** is the ratio of software time to hardware time, showing the speedup or slowdown of the hardware program. *Initialization* refers to the setup that must be done before backprojection can begin. *Backprojection* is the running of the core algorithm. *Complex Magnitude* transforms the data from complex to a real integer. Finally, *Form Image* scales the data to the range [0:255] and creates the memory buffer that is used to create the PNG image (but does not include the time required to run this API call and write the file to disk).

There are a number of significant observations that can be made from the data in Table 4.1. Most importantly, the process of running the backprojection algorithm is greatly accelerated in hardware, running over 200x faster than our software implementation. It is important to emphasize that this includes the time required to transfer projection data from the host to the FPGA, which is not required by the software program. Many of the applications discussed in Section 2.3 exhibit only modest performance gains due to the considerable amount of time spent transferring data. Here, the vast degree of fine-grained parallelism in the backprojection algorithm that can be exploited by FPGA hardware allows us to achieve excellent performance compared to a serial software implementation.

The complex magnitude operator also runs about 5x faster in hardware. In this case, the transfer of the output data from the FPGA to the host is overlapped with the computation of the complex magnitude. This commonly-used tactic allows the

data transfer time to be “hidden”, preventing it from affecting overall performance.

However, the remainder of the hardware program runs slower than the software. There are two major reasons for this. First of all, the *Initialization* step includes the time required to read the FPGA bitstream from disk and to configure the FPGA. This takes a proportionally very large amount of time; 701ms represents 50% of the total run time of the hardware application, and most (675ms, 96%) of that time is due to the time required to configure the FPGA. (The remainder is due to the allocation of large memory buffers.) This time can not be avoided, but in the Swathbuckler system it is expected that this time can be spent before the input data becomes available.

Second, the process of converting the backprojection output into an image buffer that can be converted to a PNG image (*Form Image*) runs faster in software than hardware. This step is performed in software regardless of where the backprojection algorithm was executed. The difference in run time can be attributed to memory caching. When backprojection occurs in software, the result data lies in the processor’s cache. When backprojection occurs in hardware, the result data is copied via DMA into the processor’s main memory, and must be loaded into the cache before the *Form Image* step can begin.

Table 4.2 show the single-node performance of both programs on all four data sets. Note that the reported run times are only the times required by the backprojection operation. Thus, column four, **BP Speedup**, shows the factor of speedup

Data Set	Software	Hardware	BP Speedup	App Speedup
1	24.5 s	144 ms	167.4	21.2
2	30.4 s	160 ms	179.5	25.1
3	47.7 s	263 ms	177.5	35.3
4	76.5 s	345 ms	217.6	54.4

Table 4.2: Single-node backprojection performance by data set

(software:hardware ratio) for only the backprojection operation. Column five, **App Speedup** shows the factor of speedup for the complete application including all of the steps shown in Table 4.1.

This data shows that the computation time of the backprojection algorithm is data-dependent. This is directly related to the minimum range of the projection data. According to Figure 3.2, as the subimage gets further away from the radar, the width of the radar beam is larger. This is reflected in the increased limits of the  $\chi(a, b)$  term of Equation 2.2, which are a function of the tangent of the beamwidth  $\phi$  and the range. A larger range implies more pixels are impacted by each projection, resulting in an increase in processing time. The hardware and software programs scale at approximately the same rate, which is expected since they are processing the same amount of additional data at longer ranges.

More notable is the increase in application speedup; this can be explained by considering that the remainder of the application is not data dependent and stays relatively constant as the minimum range varies. Therefore, as the range increases and the amount of data to process increases, the backprojection operation takes up a larger percentage of the run time of the entire application. For software this increase

Data Set	Errors		Max error	Mean error
1	4916	1.9%	18	1.55
2	13036	5.0%	19	1.73
3	18706	7.1%	12	1.56
4	29093	11.1%	16	1.64

Table 4.3: Image accuracy by data set

in proportion is negligible (99.5% to 99.8%) but for the hardware it is quite large (12.6% to 25.0%). As the backprojection operator takes up more of the overall run time, the relative gains from implementing it in hardware become larger, resulting in the increasing speedup numbers seen in the table.

### 4.2.2 Single Node Accuracy

Appendix A shows the result target images generated from the four data sets. Qualitatively, the hardware images look very similar, with the hardware images perhaps slightly darker near the point source target. This is due to the quantization imposed by using integer data types. When the function  $\chi(a, b)$  is computed, the floating-point software allows a slightly larger range of projection data to contribute to any given target pixel. Thus, there is a slight smearing or blurring of the point source, which is exaggerated more in software than in hardware.

Quantitatively, the two images can be compared pixel-by-pixel to determine the differences. For each data set, Table 4.3 presents the number of pixels that are different between the two images (both raw number and percentage of total pixels in the image), as well as the maximum and mean value of error. Again, errors can be attributed to the difference in the size of the window between the software and

Nodes	Software			Hardware		
	Mean	Std.Dev.	Speedup	Mean	Std.Dev.	Speedup
1	1943.2	6.15	1.0	25.0	.01	1.0
2	983.2	10.46	2.0	13.4	.02	1.9
4	496.0	4.60	3.9	7.8	.02	3.9
8	256.5	5.85	7.6	4.0	.06	6.0
16	128.4	1.28	15.1	—	—	—

Table 4.4: Multi-node experimental performance

hardware programs. For comparison, a version of each program was written that does not compute the limits of  $\chi(a, b)$  (i.e. the limits are assumed to be  $[0:\infty]$ , or an infinite beamwidth). In this case, the images are almost identical; the number of errors drops to 0.1% and the maximum error is 1. Thus, the error is not due to quantization of processed data; it remains to assume that the limit computation is responsible.

### 4.2.3 Multi-Node Performance

The second experiment involves running backprojection on multiple nodes simultaneously, using the MPI library to coordinate. This data shows how well the application scales due to *coarse-grained parallelism*, that is, the speedup that can be gained by dividing a large problems into smaller pieces and running each piece separately. For this experiment, we create an output target image that is 64 times the size of the image created by a single node. Thus, when run on one node, 64 iterations are required; for two nodes, 32 iterations are required, and so on. Table 4.4 shows the results for a single data set.

For both the software and hardware programs, five trials were run. For each trial,

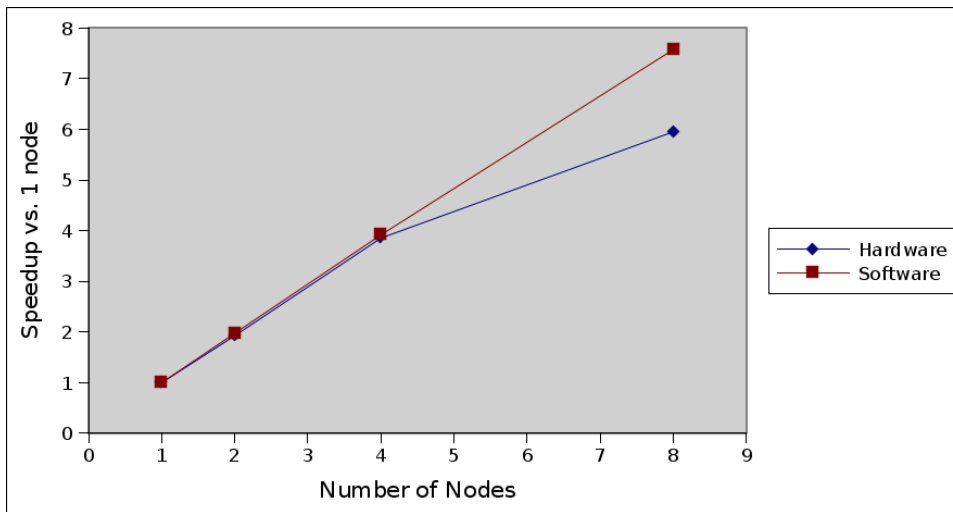


Figure 4.1: Multi-node scaling performance

the time required to run backprojection and form the resulting image on each node was measured, and the *maximum* time reported. Thus the overall run time is equal to the run time of the slowest node. The arithmetic mean of the times (in seconds) from the five trials are presented, with standard deviation. The mean run time is compared to the mean run time for one node in order to show the speedup factor.

Results are not presented for a 16-node trial of the hardware program. During our testing, it was not possible to find 16 nodes of the HHPC that were all capable of running the hardware program at once. This was due to hardware errors on some nodes, and inconsistent system software installations on others.

The speedup numbers from Table 4.4 are graphed in Figure 4.1. The linear shape of the data in the chart shows that for the backprojection application, we have achieved a nearly-ideal parallelization. This can be attributed to the lack of data passing between nodes, combined with an insignificant amount of overhead involved

Nodes	Ratio Compared To		
	1 hardware	N software	1 software
1	1.0	77.8	77.8
2	1.9	75.8	149.8
4	3.9	76.5	299.8
8	6.0	61.1	463.0

Table 4.5: Speedup factors for hardware program

in running the application in parallel with MPI. The hardware program shows a similar curve, except for  $N=8$  nodes where the speedup drops off slightly. At run times under five seconds, the MPI overhead involved in synchronizing the nodes between each processing iteration becomes significant, resulting in a slight slowdown (6x speedup compared to the ideal 8x).

The speedup provided by the hardware program is further described in Table 4.5. Compared to one node running the hardware program, we have already seen the nearly-linear speedup. Compared to an equal number of nodes running the software program, the hardware consistently performs around 75x faster. Again, for  $N=8$ , there is a slight drop off in speedup owing to the MPI overhead for short run times. Finally, we show that when compared to a single node running the software program, the combination of fine- and coarse-grained parallelism results in a very large performance gain.

### 4.3 Summary

The experimental data supports our initial hypothesis. The exceptional amount of available parallelism results in a consistently high speedup between the hardware

and software programs, and the lack of required data transfer results in good scaling behavior. The hardware program also produces accurate images that closely match the software program despite the use of integer arithmetic. Overall these results show that the backprojection algorithm is an excellent application for implementation on HPRC systems. In the next chapter we draw some conclusions from our work that can be applied to the field of HPRC, and report on possible future directions of research.

# Chapter 5

## Conclusions

The results from the previous chapter show that excellent speedup can be achieved by implementing the backprojection algorithm on an HPRC machine. As HPRC architectures improve and more applications are developed for them, designers will continue to search for ways to carve out more and more performance. Based on the lessons learned in Section 2.3 and our work on backprojection, in this chapter we suggest some directions for future research.

### 5.1 Future Backprojection Work

This project was conceived as a replacement for a portion of the Swathbuckler SAR system (see Section 3.1.1). Owing to the classified nature of that project, additional work beyond the scope of this thesis is required to integrate our backprojection implementation into that project. To determine the success of this aspect of the project, we would need to compare backprojection to the current Swathbuckler image formation algorithm, both in terms of run time as well as image quality.

Despite excellent speedup results, there are further avenues for improvement of

our hardware. The Wildstar II boards feature two identical FPGAs (see Figure 2.2), so it may be possible to process two images at once. If the data transfer to one FPGA can be overlapped with computation on the other, significant speedup is possible. It may also be possible for each FPGA to create a larger target image using more of the on-board SRAMs.

An interesting study could be performed by porting backprojection to several other HPRC systems, some of which support high-level design languages. This would be the first step towards developing a benchmark suite for testing HPRC systems; however, without significant tool support, this process would be daunting at best.

## 5.2 HPRC Systems and Applications

A common theme of Section 2.3 is data transfer. Applications that require a large amount of data to be moved back and forth between the host and the FPGA can eliminate most of the gains provided by exploiting parallelism. Backprojection does not suffer from this problem because the amount of parallelism exploited is so high that the data transfer is a relatively small portion of the run time, and some of the data transfers can be overlapped with computation. These are common and well-known techniques in HPRC application design.

This leads us to two conclusions. First, when considering porting an application to an HPRC system, it is important to consider whether the amount of available parallelism is sufficient to provide good speedup. Tools that can analyze an application to aid designers in making this decision are not generally available.

Second, it is crucial for the speed of the data transfers to be as high as possible. Early HPRC systems such as the HHPC use common bus architectures like PCI, which do not provide very high bandwidth. This limits the effectiveness of many applications. More recent systems such as the SRC-7 have included significantly higher bandwidth interconnect, leading to improved data transfer performance and increasing the number of applications that can be successfully ported. Designers of future HPRC systems must continue to focus on ways to improve the speed of these data transfers.

It is also noteworthy that the backprojection application presented here was developed using hand-coded VHDL, with some functional units from the Xilinx CoreGen library[58]. Writing applications in an HDL provides the highest amount of flexibility and customization, which generally implies the highest amount of exploited parallelism. However, HDL development time tends to be prohibitively high. Recent research has focused on creating programming languages and tools that can be used to increase programmer productivity, but applications developed with these tools have not provided speedups comparable to those of hand-coded HDL applications. The HPRC community would benefit from the continued improvement of development tools such as these.

Finally, each HPRC system has its own programming method that is generally incompatible with other systems. The standardization of programming interfaces would make the development of design tools easier and would also increase application

developer productivity when moving from one machine to the next.

### 5.3 Summary

In conclusion, we have shown that backprojection is an excellent choice for porting to an HPRC system. Through the design and implementation of this application we have explored the benefits and difficulties of HPRC systems in general and identified several important features of both these systems and applications that are candidates for porting. Backprojection is an example of the class of problems that demand larger amounts of computational resources than can be provided by desktop or single-node computers. As HPRC systems and tools mature, they will continue to help meet this demand and make new categories of problems tractable.

# Appendix A

## Experimental Data Sets

This appendix provides some additional information about the data sets that were used to test our backprojection implementations.

Four data sets were produced, each intended to produce an image that fits within the 1024x512 limit of the Target Memory SRAMs (see Section 3.3.3). These images are presented with the azimuth dimension (i.e. direction of flight) oriented along the horizontal axis, and range along the vertical axis. Conceptually, the radar dish moves from left to right across the top of the page, aimed down towards the target area.

Each data set consists of a series of targets that are a different distance away from the radar (i.e. the 'range' dimension). In the MATLAB simulation, these targets are simply point (i.e. 1x1 pixel) sources of radar reflectivity; the reflectivity value (in the range 0 to 1) is randomly assigned, along with the precise location in the target area. Table A.1 shows the minimum range between the radar dish and the target image for each data set.

Recall that Figure 3.2 shows that for a fixed beamwidth, as the range increases,

<b>Data set</b>	<b>Min. range</b>
1	250
2	500
3	1250
4	2750

Table A.1: Data set characteristics

the portion of the image affected by a single radar pulse increases. Owing to the nature of the backprojection image reconstruction algorithm, point sources will appear 'smeared' in the azimuth dimension. For data sets that correspond to larger range values, the smearing effect is increased.

The output target images are presented on the following pages. The first image shows the location of the point-source targets as input to the MATLAB script. The coordinate pair is (azimuth, range), relative to pixel (0,0) at the top-left corner of the image. Radar reflectivity of the point-source is listed as a percentage.

The second image shows the results after processing by the hardware program, and the third image the results of the software program. Refer to Section [4.2.2](#) for a discussion of the quality of the results.

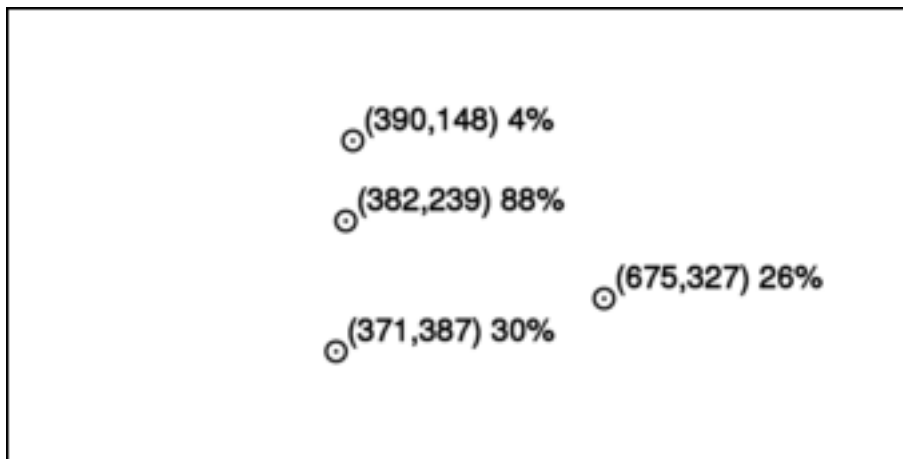


Figure A.1: Target locations, dataset 1



Figure A.2: Result image, hardware program, dataset 1



Figure A.3: Result image, software program, dataset 1

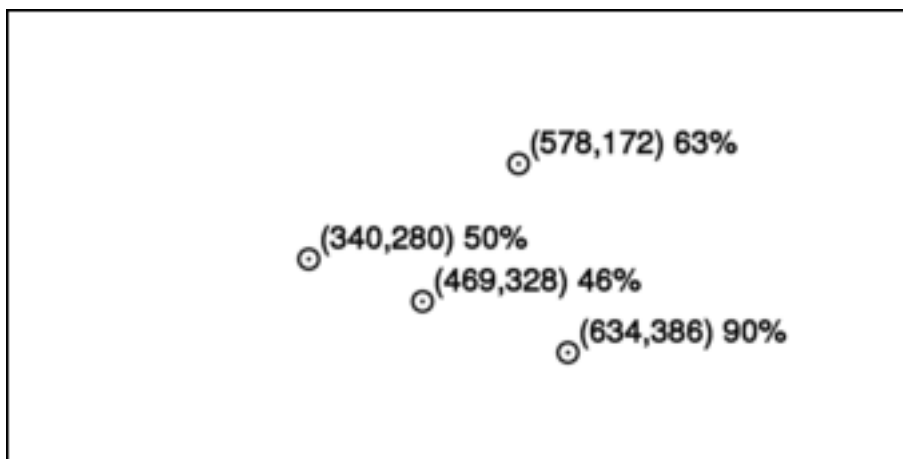


Figure A.4: Target locations, dataset 2



Figure A.5: Result image, hardware program, dataset 2



Figure A.6: Result image, software program, dataset 2

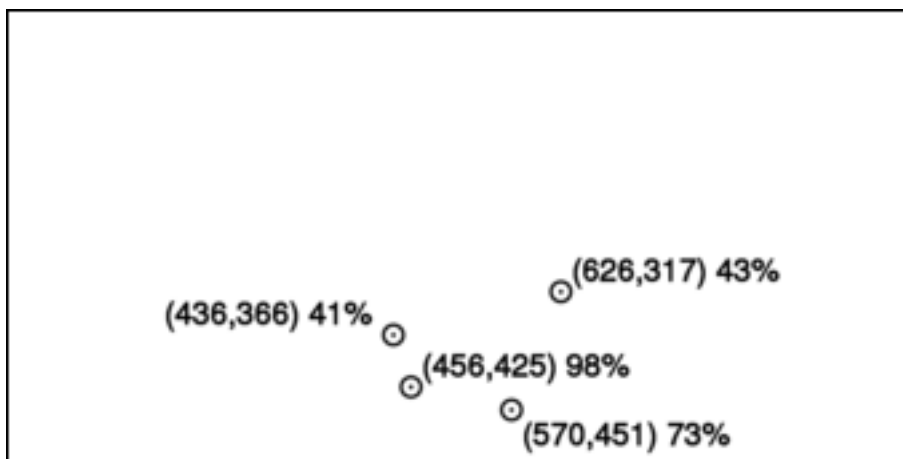


Figure A.7: Target locations, dataset 3



Figure A.8: Result image, hardware program, dataset 3



Figure A.9: Result image, software program, dataset 3

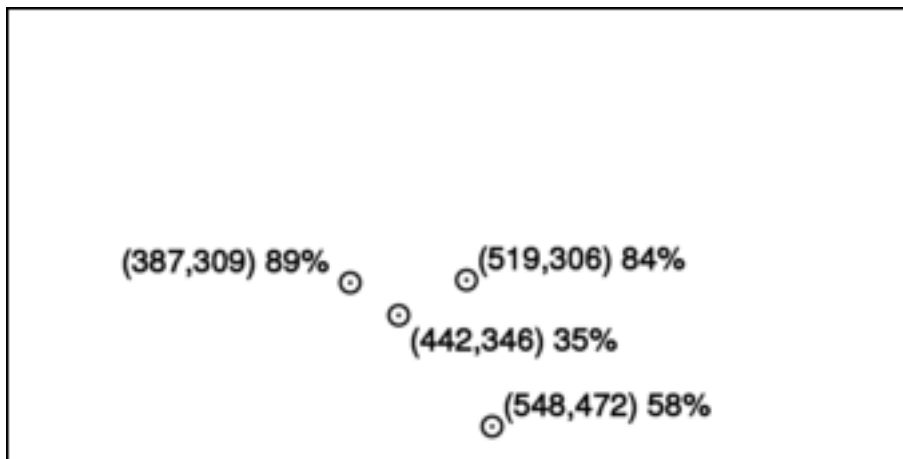


Figure A.10: Target locations, dataset 4

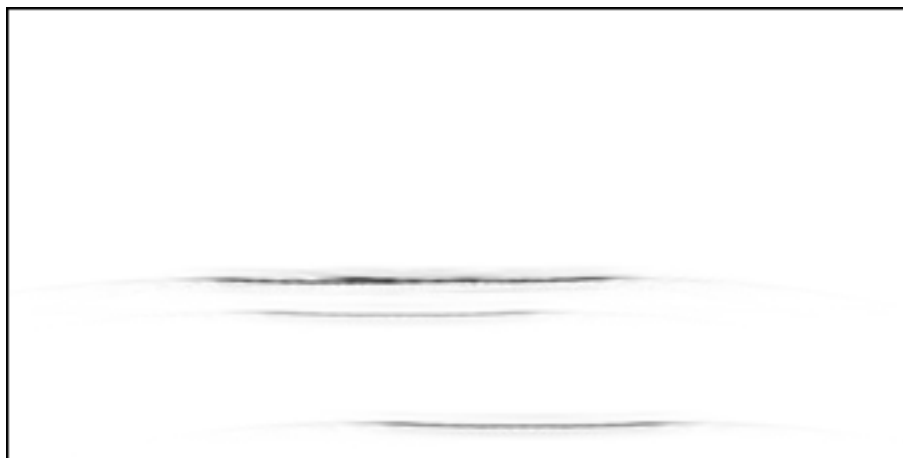


Figure A.11: Result image, hardware program, dataset 4

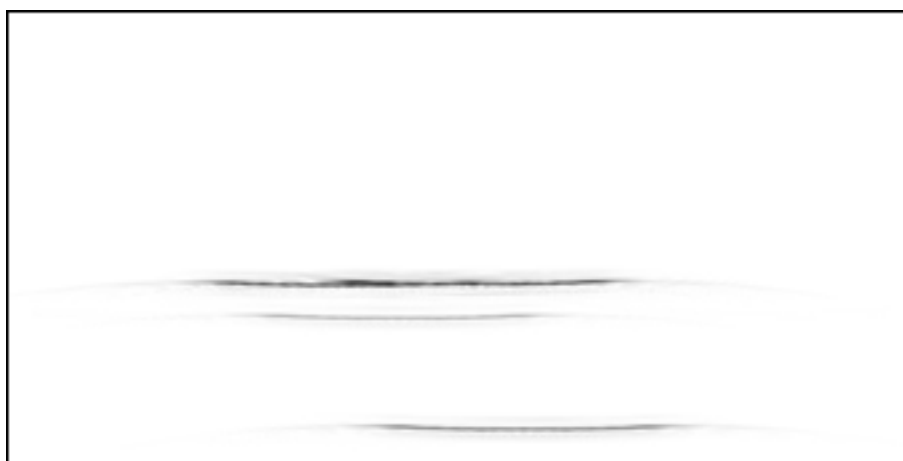


Figure A.12: Result image, software program, dataset 4

# Bibliography

- [1] A. Ahlander, H. Hellsten, K. Lind, J. Lindgren, and B. Svensson. Architectural challenges in memory-intensive, real-time image forming. In *International Conference on Parallel Processing (ICPP)*, pages 35–35, September 2007.
- [2] S. R. Alam, P. K. Agarwal, et al. Using FPGA devices to accelerate biomolecular simulations. *IEEE Computer Magazine*, 40(3):66–73, March 2007.
- [3] Annapolis Microsystems, Inc. *CoreFire FPGA Design Suite*. <http://www.annamicro.com/corefire.html>. Last accessed 3 Jan 2008.
- [4] Annapolis Microsystems, Inc. *WildStar II Data Sheet*. <http://www.annamicro.com/wsiipci.html>. Last accessed 3 Jan 2008.
- [5] Argonne National Laboratories. *MPICH*. <http://www.mcs.anl.gov/research/projects/mpich2/>. Last accessed 26 Mar 2008.
- [6] O. Bockenbach, M. Knaup, and M. Kachelrieß. Implementation of a cone-beam backprojection algorithm on the Cell Broadband Engine processor. In J. Hsieh and M. J. Flynn, editors, *Medical Imaging 2007: Physics of Medical Imaging*, volume 6510 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, March 2007.
- [7] U. Bondhugula, A. Devulapalli, J. Dinan, J. Fernando, P. Wyckoff, E. Stahlberg, and P. Sadayappan. Hardware/software integration for FPGA-based all-pairs shortest-paths. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 152–164, 2006.
- [8] U. Bondhugula, A. Devulapalli, J. Fernando, P. Wyckoff, and P. Sadayappan. Parallel FPGA-based all-pairs shortest-paths in a directed graph. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.
- [9] D. A. Buell, S. Akella, J. P. Davis, E. A. Michalski, and G. Quan. The DARPA boolean equation benchmark on a reconfigurable computer. In *Military and Aerospace Programmable Logic Devices (MAPLD)*, 2004.

- [10] Celoxica Ltd. *DK Design Suite*. <http://www.celoxica.com/products/dk/>. Last accessed 3 Jan 2008.
- [11] A. Conti, B. Cordes, M. Leeser, E. Miller, and R. Linderman. Adapting parallel backprojection to an FPGA enhanced distributed computing environment. In *Workshop on High-Performance Embedded Computing (HPEC)*, September 2005.
- [12] B. Cordes, A. Conti, M. Leeser, and E. Miller. Performance tuning on reconfigurable supercomputers. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing (SC06)*, November 2006. Poster session.
- [13] B. Cordes, M. Leeser, E. Miller, and R. Linderman. Improving the performance of parallel backprojection on a reconfigurable supercomputer. In *Workshop on High-Performance Embedded Computing (HPEC)*, September 2006. Poster session.
- [14] S. Coric, M. Leeser, E. Miller, and M. Trepanier. Parallel-beam backprojection: an FPGA implementation optimized for medical imaging. In *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 217–226, February 2002.
- [15] Cray Inc. *Cray XD1 Supercomputer*. <http://www.cray.com/products/xd1/>. Last accessed 20 Dec 2006.
- [16] Cray Inc. *Cray XT5h Supercomputer*. <http://www.cray.com/products/xt5/xt5h.html>. Last accessed 3 Jan 2008.
- [17] DSPLogic Inc. *Reconfigurable Computing Toolbox*. <http://www.dsplogic.com/home/products/rctb>. Last accessed 3 Jan 2008.
- [18] T. El-Ghazawi, E. El-Araby, A. Agarwal, J. LeMoigne, and K. Gaj. Wavelet spectral dimension reduction of hyperspectral imagery on a reconfigurable computer. In *Military and Aerospace Programmable Logic Devices (MAPLD)*, 2004.
- [19] P. Eugster, P. Felber, R. Guerraoui, and A. M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.
- [20] N. Gac, S. Mancini, and M. Desvignes. Hardware/software 2d-3d backprojection on a SoPC platform. In *ACM Symposium on Applied Computing (SAC)*, pages 222–228, New York, NY, USA, 2006.
- [21] M. Gokhale, C. Rickett, J. L. Tripp, C. Hsu, and R. Scrofano. Promises and pitfalls of reconfigurable supercomputing. In *International Conference on Engineering of Reconfigurable Systems & Algorithms (ERSA)*, pages 11–20, 2006.

- [22] *GPGPU: General Purpose Computation Using Graphics Hardware*. <http://gpgpu.org/>. Last accessed 3 Jan 2008.
- [23] *GTK+ Project*. <http://gtk.org/>. Last accessed 5 Mar 2008.
- [24] A. Hast and L. Johansson. Fast factorized back-projection in an FPGA. Master's thesis, Halmstad University, 2006. <http://hdl.handle.net/2082/576>.
- [25] M. C. Herbordt, T. VanCourt, Y. Gu, B. Sukhwani, A. Conti, J. Model, and D. DiSabello. Achieving high performance with FPGA-based computing. *IEEE Computer Magazine*, 40(3):50–57, March 2007.
- [26] High Performance Technologies Inc. *Cluster Computing*. <http://www.hpti.com/tblTopicsHomeTemplate.asp?id=109>. Last accessed 3 Jan 2008.
- [27] IBM. *Cell Broadband Engine*. <http://www.ibm.com/technology/cell/>. Last accessed 3 Jan 2008.
- [28] Impulse Accelerated Technologies Inc. *Impulse Products*. [http://www.impulsec.com/fpga\\_c\\_products.htm](http://www.impulsec.com/fpga_c_products.htm). Last accessed 3 Jan 2008.
- [29] A. C. Kak and M. Slaney. *Principles of Computerized Tomographic Imaging*. IEEE Press, New York, 1988.
- [30] R. W. Linderman. Swathbuckler: Wide swath SAR system architecture. In *IEEE Conference on Radar*, April 2006.
- [31] Mercury Computer Systems Inc. *FCN Developers Kit*. <http://www.mc.com/products/productdetail.aspx?id=2816>. Last accessed 3 Jan 2008.
- [32] Mercury Computer Systems Inc. *MCJ6 FCN Module*. <http://www.mc.com/products/productdetail.aspx?id=2624>. Last accessed 3 Jan 2008.
- [33] Mercury Computer Systems Inc. *Multicomputer Operating Environment*. <http://www.mc.com/products/productdetail.aspx?id=2820>. Last accessed 3 Jan 2008.
- [34] Mercury Computer Systems Inc. *Powerstream 7000 FCN Module*. <http://www.mc.com/products/productdetail.aspx?id=2724>. Last accessed 3 Jan 2008.
- [35] Mercury Computer Systems Inc. *Powerstream 7000 System*. <http://www.mc.com/products/productdetail.aspx?id=2878>. Last accessed 3 Jan 2008.
- [36] Mercury Computer Systems Inc. *RACE++ 6U VME System*. <http://www.mc.com/products/productdetail.aspx?id=2856>. Last accessed 3 Jan 2008.

- [37] J. S. Meredith, S. R. Alam, and J. S. Vetter. Analysis of a computational biology simulation technique on emerging processing architectures. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–8, 2007.
- [38] Mitrionics. *Mitrion SDK*. <http://www.mitrion.com/default.asp?pId=23>. Last accessed 3 Jan 2008.
- [39] N. Moore, A. Conti, M. Leeser, and L. S. King. Vforce: An extensible framework for reconfigurable supercomputing. *IEEE Computer Magazine*, 40(3):39–49, March 2007.
- [40] G. R. Morris, V. K. Prasanna, and R. D. Anderson. A hybrid approach for mapping conjugate gradient onto an FPGA-augmented reconfigurable supercomputer. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 3–12, 2006.
- [41] R. A. Neri-Calderon, S. Alcaraz-Corona, and R. M. Rodriguez-Dagnino. Cache-optimized implementation of the filtered backprojection algorithm on a digital signal processor. *Journal of Electronic Imaging*, 16(4), 2007.
- [42] L. Nguyen, M. Ressler, D. Wong, and M. Soumekh. Enhancement of backprojection SAR imagery using digital spotlighting preprocessing. In *Proceedings of the IEEE Radar Conference*, pages 53–58, April 2004.
- [43] OpenFPGA. <http://www.openfpga.org>. Last accessed 3 Jan 2008.
- [44] V. Ross. Heterogeneous high performance computer. In *HPCMP Users Group Conference*, pages 304–307, 2005.
- [45] S. Rouse, D. Bosworth, and A. Jackson. Swathbuckler: Wide array SAR processing front end. In *IEEE Conference on Radar*, April 2006.
- [46] R. Sass, W. V. Kritikos, A. G. Schmidt, S. Beeravolu, P. Beeraka, K. Datta, D. Andrews, R. S. Miller, and D. Stanzione Jr. Reconfigurable computing cluster (RCC) project: Investigating the feasibility of FPGA-based petascale computing. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 127–140, 2007.
- [47] R. Scrofano and V. K. Prasanna. Preliminary investigation of advanced electrostatics in molecular dynamics on reconfigurable computers. In *Proceedings of the ACM/IEEE Supercomputing 2006 Conference (SC06)*, 2006.
- [48] Silicon Graphics Inc. *SGI RASC Technology*. <http://www.sgi.com/products/rasc/>. Last accessed 3 Jan 2008.

- [49] M. C. Smith, J. S. Vetter, and S. R. Alam. Scientific computing beyond CPUs: FPGA implementations of common scientific kernels. In *Military and Aerospace Programmable Logic Devices (MAPLD)*, 2005.
- [50] M. C. Smith, J. S. Vetter, and X. Liang. Accelerating scientific applications with the SRC-6 reconfigurable computer: Methodologies and analysis. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2005.
- [51] M. Soumekh. *Synthetic Aperture Radar Signal Processing with MATLAB Algorithms*. John Wiley and Sons, New York, 1999.
- [52] SRC Computers Inc. *CARTE Programming Environment*. <http://www.srccomp.com/techpubs/carte.asp>. Last accessed 3 Jan 2008.
- [53] SRC Computers Inc. *SRC-7*. <http://www.srccomp.com/products/src7.asp>. Last accessed 3 Jan 2008.
- [54] J. Tripp, M. B. Gokhale, and A. A. Hansson. A case study of hardware/software partitioning of traffic simulation on the Cray XD1. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(1):66–74, January 2008.
- [55] S. Tucker, R. Vienneau, J. Corner, and R. W. Linderman. Swathbuckler: HPC processing and information exploitation. In *IEEE Conference on Radar*, April 2006.
- [56] K. D. Underwood, K. S. Hemmert, and C. Ulmer. Architectures and APIs: Assessing requirements for delivering FPGA performance to applications. In *Proceedings of the ACM/IEEE Supercomputing 2006 Conference (SC06)*, 2006.
- [57] X. Wang, S. Braganza, and M. Leeser. Advanced components in the variable precision floating-point library. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 249–258, April 2006.
- [58] Xilinx, Inc. *CORE Generator*. [http://www.xilinx.com/products/design\\_tools/logic\\_design/design\\_entry/coregenerator.htm](http://www.xilinx.com/products/design_tools/logic_design/design_entry/coregenerator.htm). Last accessed 4 Mar 2008.
- [59] Xilinx, Inc. *Virtex-II Data Sheet*. [http://www.xilinx.com/products/silicon\\_solutions/fpgas/virtex/virtex\\_ii\\_platform\\_fpgas/](http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex_ii_platform_fpgas/). Last accessed 3 Jan 2008.
- [60] Xilinx, Inc. *Virtex-II Pro Data Sheet*. [http://www.xilinx.com/products/silicon\\_solutions/fpgas/virtex/virtex\\_ii\\_pro\\_fpgas/](http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex_ii_pro_fpgas/). Last accessed 3 Jan 2008.

- [61] X. Xue, A. Cheryauka, and D. Tubbs. Acceleration of fluoro-CT reconstruction for a mobile C-arm on GPU and FPGA hardware: A simulation study. In M. J. Flynn and J. Hsieh, editors, *Medical Imaging 2006: Physics of Medical Imaging*, volume 6142 of *Proceedings of the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 1494–1501, March 2006.
- [62] L. Zhuo and V. K. Prasanna. High performance linear algebra operations on reconfigurable systems. In *Proceedings of the ACM/IEEE Supercomputing 2005 Conference (SC05)*, page 2, Washington, DC, USA, 2005. IEEE Computer Society.